

Computer Science Department

TECHNICAL REPORT

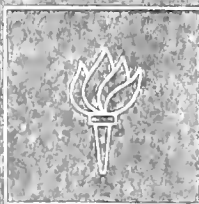
A Theory of Natural Learning

Alexander Botta

Technical Report 560

May 1991

NEW YORK UNIVERSITY



Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012

NYU COMPSCI TR-560
Botta, Alexander
A theory of natural
learning.

6.1



NEW YORK UNIVERSITY
COMPUTER SCIENCE LIBRARY
201 May 1991 New York, NY 10012

A Theory of Natural Learning

Alexander Botta

Technical Report 560

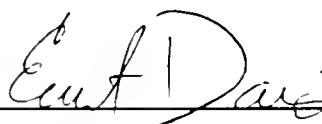
May 1991

A Theory of Natural Learning

Alexander Botta

May 1991

A dissertation in the Department of Computer Science submitted to the
faculty of the Graduate School of Arts and Science in partial fulfillment
of the requirements for a degree of Doctor of Philosophy
at New York University

Approved: _____

Professor Ernest Davis, Advisor

This document reproduces the author's Ph.D. Dissertation.

©Copyright by Alexander Botta, 1991

ALL RIGHTS RESERVED

Abstract

Unsupervised learning is based on capturing regularities in data. We formalize the vague notion of regularity using the concept of algorithmic information (Solomonoff, Chaitin, Koppel). We present a theory on how regularities are induced and accumulated. A generative model captures a *regularity* if it achieves compression. A *basic regularity* is a building block for hierarchical structures. We prove that a basic regularity may be identified as a local maximum in compressibility as a function of model complexity. *Stepwise induction* is a polynomial time approach to structures whose basic components have bound complexity.

Agents engage in *active learning*. The regularities of their sensory-motor streams are similar to Piaget's schemes and constituents of an induced ontology. We illustrate these ideas on three microworlds. First are Moore automata. State representations are constructed incrementally from the results of tests performed when in that state, and from sequences of outputs perceived on the way to that state. The second world contains loosely coupled geometric objects. They are *basic regularities* identifiable by *stepwise induction*. In the third world agent has an elaborate eye and can move objects on a tiled surface. Statistical correlations between sets of stimuli are induced, then models are constructed to generate instances of new correlations from already known ones.

Algorithmic information theory allows a unified perspective on many areas of learning research. We define *analysis* as the separation of the novelty in data from the already known. We present explanation based generalization as a well formalized instance of analysis and constructive induction is an ill defined instance. We show EBG to specialize a theory through positive examples, and prove it a language independent method, valid beyond the predicate calculus representations.

CONTENT

Acknowledgments	vi
Organization (Extended Abstract)	vii
1. Themes	1
2. Learning as characterization of structure	8
2.1 Induction Based on Algorithmic Probability	8
2.2 Algorithmic Complexity and the Quantity of Information	10
2.3 Structure and Randomness	14
2.4 The Minimum Description Length (MDL) Principle	18
- Bayesian Motivation	18
- Approximations for Algorithmic Complexity	19
2.5 Unification of Learning and Problem Solving	22
3. Creation of elements for a private language	29
3.1 Representations and Regularities	32
3.2 Models and Codes	34
3.3 Induction by Refinement	39
3.4 Finite Memory Models	44
- The Class of Models	45
- Running and Parsing	45
- Refinement	47
- Incremental Computation of Compressibility	50
- Computation of Transition Schemata	50
- Composition	51
3.5 Induction of Submodels: Libraries	53
- The Class of Models	53
- Fragmentation, Parsing and Refinement	56
3.6 Stepwise Induction	59

4. Learning an environment by exploration	71
4.1 A Psychological Perspective	71
- A Semantic Basis for Learning Language	72
- The Agent Living in a Passive, Deterministic Environment	75
- Some Future Themes	79
4.2 Model World 1 (Induction of Moore automata)	80
- Characterization of States by Tests	81
- Characterization of States by Id_sequences	84
4.3 Model World 2 (Objects and Simple Eye)	87
- The Class of Domains	87
- Objects as Equivalence Classes of Substructures	90
- A Practical Exercise	95
4.4 Model World 3 (Tiles-land)	100
- The Class of Domains	100
- Building New Representations	
Based on Co-occurrence of Attributes	104
- Induction Scenario	109
5. Background Knowledge and The Uses of Memory	117
5.1 The Learning Cycle	117
5.2 Explanation Based Learning	122
- The Proof As A Code	122
- Generalization of Explanation by Retaining its Structure	123
- Specialization of Theory Through Positive Examples	124
5.3 Constructive Induction	127
5.4 Bongard Problems	131
6. Discussion and Research Directions	137
Bibliography	143

Acknowledgments

First of all, I owe special thanks to Ernest Davis whose relentless critiques turned a confused babble into this text. Other people helped me on the way and I would like to express my gratitude to them. Malcolm Harrison gave me long ago a copy of Solomonoff's 1964 papers and called my attention to this research direction. Dana Angluin listened patiently to my initial incoherent intentions. The presentation that M. Koppel gave at the N.Y.U contributed a lot to my finding a clear direction. Conversations with Pasquale Caianiello, whose interests are close to mine, provided courage and precious insights. Leo Joskowicz and Roken Ahmed imparted to me a necessary concern for practicality. Leo Joskowicz also helped me make this text intelligible. In spite of the normal usage of acknowledgments, I cannot help feeling gratitude for all those authors whose papers and books guided me to this point. Finally, I owe special thanks to my friends Tamara Botta, Dan Tomescu, Adrian Nachman and Robert High who contributed in important ways to my determination.

Organization (Extended Abstract)

This is a computational theory about simulated agents that acquire knowledge by interacting with well defined, environments. We call it *natural learning* because we focus on those problem features that makes it relevant for understanding unsupervised concept formation in live beings. We use the framework of algorithmic information theory to define *regularity* and *natural spaces*: structures that accept a decomposition in recurrent, recognizable *regularities*. We focus on several sub-topics: construction of new representations, minimal length encoding, incremental accumulation of partial knowledge, abilities required by natural spaces, the semantic basis for language learning, the role of background knowledge. **Chapter 1** introduces each of these main themes of natural learning in an informal way.

Learning is grasping constraints in data, hence it is equivalent to data compression. We use the concepts of minimal length encoding and algorithmic complexity because they provide a representation independent approach to measuring compressibility. **Chapter 2** reviews the main concepts from algorithmic information theory [Solomonoff-64][Chaitin-78]. It presents a distinction between the structure and randomness in data [Koppel-88]. It gives a standard Bayesian motivation for the minimum description length principle. It reviews a few computable approximations for the algorithmic complexity. It mentions some directions that we shall follow in using these concepts. Section 2.5 contains our contributions to a unification of learning and problem solving [Solomonoff-86]. We outline some of its difficulties and interesting consequences. One is the distinction between learning to solve problems and learning to learn.

Chapter 3 introduces our main definitions, all revolving around the concept of *regularity* (complete, partial and basic). A generative model captures a regularity if it achieves a compressed representation. We investigate several induction algorithms for capturing regularities. We prove a convergence result and a link between compression and prediction. We give an incremental induction algorithm for Finite Automata as representations of regularity. We define Finite Memory Models as a way to increase the representational power of Finite Automata while retaining their simplicity of induction. We also define *libraries*, a class of regularities that permits accumulation of partial knowledge. We show it to be a generalization of clustering. Following [Caianello-71], we introduce the concept of *basic regularity* as a building block for hierarchical structures. We show how it can be identified as a local maximum in compressibility as a function of model complexity. We define *stepwise induction*. It consists in finding a model, using it to compress the data, then applying the same procedure on the code. It is a way to induce, in polynomial time, structures whose basic components have bound complexity. We work out a few practical examples of induction of hierarchical structures.

Chapter 4 focuses on simulated active agents exploring an environment and acquiring sensory-motor skills. An agent induces regularities of its sensory-motor stream, that is, the flow of actions and perceptions. We link these regularities to Piaget's schemes. By capturing them, the agent attaches meaning to symbolic terms and constructs an ontology of its universe. This approach should allow the proverbial blind men to put together an accurate representation of their elephant. We illustrate these ideas on three classes of microworlds of increasing complexity.

Class 1 are Moore automata experienced by the agent through their inputs and outputs. We present two induction approaches for incremental construction of state representations. One identifies a state by the results of tests performed by the agent when in that state. The other identifies a state by the sequence of outputs perceived by the agent on the way to reach that state. They both work through accumulation and refinement of partial knowledge. *Class 2* contains simple objects that are, with a few exceptions, independently affected by the agent's actions. It still accepts a Moore automaton model but the representations constructed during the exploration are more complex. Objects are *basic regularities* of the environment structure identifiable by *stepwise induction*. We apply here the classes of models developed before. In *Class 3*, the agent has an elaborate eye and can move objects on a tiled surface. We investigate here a possible explanation for how complex representations may emerge as computational links between a new data item and prototypes already present in memory. We sketch a two layered induction scenario for certain regularities of this world. A first layer detects and stores statistical correlations between sets of stimuli. A second layer constructs models that can generate instances of new correlations from already known ones, stored in memory. This layer can achieve shorter, computationally complex representations for correlations. We show that the emergent representations are compact and adequate for certain classes of events of this world.

Chapter 5 defines *analysis* as the separation of the novelty in data from the part that is already known from background information. We relate analysis to models, codes and to hypothesis refinement. We identify it in several standard areas of learning research and discuss in more details two extreme cases: the tightly formalized explanation based learning and the loosely formalized constructive induction. We show EBG to be able to use positive examples to specialize a theory, and we prove it a language independent method, valid beyond the predicate calculus representations. We investigate how background knowledge can be used to guide the construction of appropriate representations. We work out an example of Bongard Classifications.

Chapter 6 summarizes our results and points to possible continuations. **Chapters 1,2 and 3** are prerequisites for **4** and **5**. The text contains many examples clearly framed. Those accompanied by bibliographic references present fragments of previous research that support the concepts developed here. The others present our own experiments.

1. Themes

Learning is an attribute of life. Only by forced analogy we can say that the sand learned the shape of a foot. [Maturana-86] claims that *all living systems are cognitive systems and to live is to know*. One of the metaphors he uses for life is “a continuous drift on the sea of change while maintaining the shape of the ship”. Living organisms have to reflect in their structure some predictable component of their environment in order to maintain their life. Life involves accumulation of knowledge both by evolution of structures over generations, and by an individual over a lifetime.

Machine learning research focussed on carefully carved fragments of the large problem of understanding learning. Strong assumptions about what a learner needs to know a priori in order to learn new things, resulted in algorithms able of impressive performance in narrow situations (For example PAC algorithms [Valiant-84], decision trees [Quinlan-89], etc.). However, approaches that require few assumptions - known as *weak learning* - seem to us a better starting point for explaining learning in a large variety of situations. Learning in living organisms is certainly focussed on some aspects of the environment: those that matter to them. Innate perceptive mechanisms make bacteria, insects and birds concentrate on different things. Still a certain flexibility remains. Understanding this flexibility was our main motive for constructing a computational model of *natural learning*. Also, studies of seemingly simple learning problems show them to be computationally intractable. Then, how is learning possible at all? We argue that natural learning should be concerned with the structure of those spaces that accept a decomposition in recurrent, recognizable parts. Expressing the whole depends on being able to name the parts. For example, the syntax of a sentence is based on the intermediate concept of word. The concept of rectangle illustrated by vectors of pixels depends on the intermediate concepts of pixel adjacency and line segment. Such structures are usually hierarchical, involving several levels of vocabularies. They suggest learning at first a dictionary of building blocks, and only then concentrating on how they are assembled. We call them *natural spaces*. We shall argue now about what aspects are relevant for a definition of natural learning:

(a) *Construction of adequate representations*. Many algorithmic approaches to learning assume a given language to express both the training material and the result of the learning process. This is a well known limitation and there have been several attempts to overcome it. Typical names for this are *bias shifting* [Utgoff-84] and *constructive induction* [Michalski-83],[Rendell-85,86,89]. So, *one challenge is to describe a learning strategy that adapts its bias*. That is, how to create a language adequate for grasping the structures that have to be learned without assuming an a priori knowledge about those structures. A learner should develop its own concepts that are appropriate at organizing the apparently amorphous data offered by experience. One can make an analogy to the development of subroutines for a certain programming job, or of programming cliches adequate for a certain programming environment.

(b) *Compression*. An infinite stream of experience can only be concisely expressed if the learner discovers regularities (redundancy) in it. An alternate view is to discover that not everything is possible: to a certain extent any *learning consists in discovery of constraints*. Therefore regularities are constraints. But any regularity present in a set of observations can be used to obtain a compressed description of them. This intuition supports the equivalence between learning and the problem of minimal length encoding. An adequate language is one that is able to describe a possibly infinite reality in concise expressions.

If , for example, the proper language to describe regularities in arrays of pixels should contain a term for *circle*, then the learner must possess an algorithm to recognize the presence of circles. In other words, such a language must not depend on some exterior interpretation. *The semantics of terms and combinators must be grounded in the perceptions and innate computational abilities of the learner.*

Example 1.1 [Wolff-82,87] is one of the early examples of induction guided by a concern for compressed representations. He uses size of grammars to guide a bottom-up induction of a context free grammar for a corpus of natural language sentences. His program SNPR finds common substrings in the text, then strings that are interchangeable in a given context, then recursive rules, and gradually builds a minimal size grammar for the corpus.

Example 1.2 [Holder-89] is concerned with induction from complex, structured examples like organic chemistry formulae. He finds that traditional induction algorithms work better on examples that were modified by compression. Therefore he searches for repeating subgraphs in a structured organic chemistry formula. The search algorithm attempts to maximizes 4 parameters

- compression factor (substructure complexity times number of occurrences)
- substructure density (number of links vs. number of nodes),
- isolation of substructure in ensemble (how strongly clustered),
- coverage factor (how much of the formula is accounted for by instances of the substructure)

The implicit result of this is the fact that important substructures (like benzene rings) become elements for a new representation. Also, this suggests that there is a trade-off between the size of a substructure and its utility in a compressed representation.



(c) *Active learning through environment exploration.* Natural learning occurs to an agent through exploration of its habitat. Such an agent is active: not only does it process data presented to it, but it also acts on its universe, affecting the order of data presentation. Actions and partial views of the universe available through its senses will mix in a continuous stream. Regularities of this stream will reflect the possible intricate ways in which the agent and its environment are acting on each other. Computer simulations of such settings allow a clear statement of the learning problem: How much of the environment structure will eventually be perceived by an agent endowed with the given sensors and the given actions.

The agent/environment setting allows a simple model for constructing a representation together with its interpretation. New symbols attached to discovered regularities are grounded into the agent's experience. They do not need to be associated to some exterior domain, they only name predictable dependencies between actions and results.

(d) *Incremental accumulation of partial knowledge.* Natural learners learn continuously. Certainly, there may be worlds where a surprise is always possible that may invalidate all concepts built so far. That is, all previous experience is lost because it was interpreted and retained in a certain way. Even incremental induction of structures as simple as finite automata encounters this problem. Therefore it is important to characterize the structures that are learnable incrementally and develop ways to accumulate knowledge that doesn't have to be completely discarded at a first counterexample.

An agent exploring an universe will have to learn incrementally. In an infinite universe, any theory is potentially refutable, yet a rational agent has to find some territories of predictability. Instead of producing a final outcome, the learning process would have to gather a set of current beliefs about the universe and refine them to accommodate the new experience. Libraries, introduced in chapter 3, provide a representation for such unrelated fragments of experience. We shall apply this idea in several toy worlds where we can sketch a learning strategy for a rational agent.

(e) *Learning abilities required by natural spaces.* In most classes of induction problems we can find hard instances that make the class intractable. We do not think that learner's abilities must cover those limit cases. However the learner should be able to grasp natural situations and their intricacies. We assume that, though complex, the natural spaces contain fundamental regularities and many of them display a hierarchical structure. Therefore we expect partial theories to have a fair chance to contribute to more complex, better theories because they reflect repetitive building blocks of the environment itself.

A large family of physical domains have such a structure, at least in a naive, sim-

plified view, so they deserve the name of *natural spaces*. As a consequence, we shall accept an agent, confronted with such spaces, to be utterly confused at some points of its learning process, but to be able to make some rational predictions at some other points. The word *rational* should be understood as based on experience.

Example 1.3 Figures 1.a, 1.b, 1.c show three instances of a raster with some points marked 'x'. We are trying to illustrate the concept of *straight line segment*. Let us imagine these instances as part of an infinite string of pictures, each showing one horizontal or vertical segment of x's.

Figure 1: positive examples for the concept <i>straight line segment</i>		
1.a	1.b	1.c
- - - - -	- - - - -	- - - - - x -
- - x - - - -	- - x x x - -	- - - - - x -
- - x - - - -	- - - - - - -	- - - - - x -
- - x - - - -	- - - - - - -	- - - - - x -
- - x - - - -	- - - - - - -	- - - - - x -

We can certainly *see* these figure, but, for the purpose of symbolic processing, we have to specify a symbolic description language for these examples. Let us say that they are given as sets of values for 35 boolean variables (one for each position of the raster). The illustrated concept, *straight line segment*, can be easily expressed as a boolean function of these variables, but its usual representations likely to be produced by today's specialized algorithms (CNF, DNF, decision tree, circuit) have, from our point of view, considerable drawbacks.

First, these expressions are quite long. The learner may not have the resources to induce them. Then, they are not easily extendable to other *related* concepts (for example: rectangle, oblique line segment). Those would have to be induced separately. We could forgive a natural learner not having the necessary resources for the induction of such a boolean function. However, we would like it to notice simpler, partial regularities, like

- (1) *there is a stable neighborhood relation between variables such that if one is true, one of its neighbors is also true, or*
- (2) *the number of x's is between 2 and 7.*

Forging a representation for (1) facilitates full induction of *line segment* and other related concepts. Here is how. Let us say that neighborhood relations, a regularity of

this domain, are assimilated as partial knowledge, and they contribute to a new representation. In figure 1.a the position of the lowest 'x' is denoted by the constant 'a'. A possible description of the set of points marked 'x' is the following Lisp expression that makes use of the *neighbor-of* function *up*:

$$d_1 = (a (up a) (up (up a)) (up (up (up a))))$$

We would like the learner to perceive the common structure of these expressions. For example, the one above can be expressed as

$$d_2 = (f 'a 'up 4)$$

where *f* is a simple recursive function. Note that, using *f*, all pictures can be succinctly represented as lists of four elements. Besides, while the new representation makes the common element of those pictures explicit, by attaching the symbol *f* to it, it also distinguishes their random parts, that is their parameters. Indeed, now we can attach English names to symbols, and interpret d_2 as

$$(\text{line-segment} \quad \text{starting-point} \quad \text{direction} \quad \text{length})$$

The new, constructed representation achieves both minimal encoding and visibility of essential features.



(f) *Memory, analysis, and background knowledge.* Going back to the desirable characteristics of natural learning, we would like to elucidate how new knowledge is integrated into experience. Specifically we are interested in how we can organize a memory that stores prototypes of regularities, in order to answer the following question. *How much of the new data presented to the learner is an instance of known regularities and how much is really new?* Analysis is this process of discrimination.

We shall also look at what classes of structures allow decomposition into partial, independent regularities. More precise definitions of *regularity* and *rational prediction* will follow. Even if background knowledge is not given to the learner, once some regularity is learned, it constitute a background for further learning. Based on this view, we shall argue that the traditional distinction between Explanation-Based and Similarity-Based learning is superficial.

(g) *Creating the semantic basis for learning natural language.* The descriptive and communicative power of a few hundred words in a natural language is possible because those words reflect ubiquitous substructures of the physical universe. Therefore,

in order to learn a natural language, a learning agent should be able to construct a simple ontology of its universe.

Let us assume that we have an algorithm able to capture the syntax of a natural language. The proper usage of the language still depends on its semantics, that is on the situations the sentences refer to. An agent learning the language has no direct access to those situations but it has access to accumulated regularities of its actions / sensors interaction. These regularities may then serve as a basis for learning a language that *speaks* about the same experience.

The agent / environment setting may also offer a fertile context to build computational models for certain psychological issues. Piaget's constructive theory of sensory-motor intelligence is one of them. How does internalization of experience happen? Can physical exploration offer support for mental exploration? Actions taken by the agent will cause certain transformations of the universe. These will be reflected as a transformations among perceived sensory images. Could these transformations be further abstracted, stored and later used as new operators on representations?



Let us use the above themes in making a few remarks on some established categories of machine learning research.

(i) *Rote*. To start with the obvious, tape recorders display a very primitive form of learning. Haven't we all been accused in some first grade classroom that we hadn't *understood* the subject but we had only *learned it by heart*? Simple memorization is limited by the availability of memory and by the fact that fragments learned by rote are not related to each other or to background information and therefore are not useful except in situations identical to those where they were learned.

(ii) *Induction* usually amounts to finding a function (program) able to output the given sequence and the techniques are mostly enumerative. A hypothesis that fails is discarded even if it matched some part of the sequence. If this process is seen to be the development of a language, than partial models that explained a large enough number of subsequences, should be retained as part of a learned vocabulary.

(iii) *Concept learning* assumes that examples are expressed in a language that makes all relevant features explicit. A quick look at the problems proposed in [Bongard-75] (all are about learning a classification procedure from six positive and six negative examples) makes obvious how difficult it is to specify a class of functions from which the target concept should be chosen. The main reason is the fact that examples are not presented in a language that is proper to expressing the concepts. Besides, concept learning seems to be didactically abstracted from the natural problem of

a being that explores an environment. There may be more than one concept to learn in order to represent the environment in a concise way.

(iv) *Clustering* and other unsupervised learning methods are closer to our approach. They create categories to explain the data and there are good incremental clustering algorithms. The resulting concepts are only sets and they don't relate to each other except through inclusion. However it suggests the idea of accumulation of knowledge in small fragments.

(v) *Maximization of Performance Function*. Improving the performance on some activity (search through some adequate space) needs reasoning on previous instances of that activity to discover useful particularities of that space. Naming important regions of that space and relationships among them is a way to discover its essential structure. STRIPS macrooperators or SOAR chunks are example of this approach. Building better plans, search paths, etc. can be done only by taking into account that structure.

(vi) *Discovery* includes so far two kinds of attempts of building a computational model for the scientific activity. One is finding symbolic rules to account for a lot of experimental data [Langley-87,89], the other one is a sort of constructive mathematics guided by some predefined preferences for concepts or properties considered *interesting* [Haase-86,87][Lenat-78,83,84]. We think that this latter process would be better guided by the need to construct appropriate representations during the exploration of some practical domain. That is, while attempting to describe a domain might be called *physics*, forging the appropriate symbolic representations, and studying the way the representations can be changed into equivalent, better ones might be called the *mathematics* of that domain and should be classified as *discovery*.

In conclusion, we propose to focus not so much on finding one function or formula that accounts for the data but on finding a set of subroutines that might serve as building blocks for theories that account for a potentially infinite natural space. The concept of regularity will be our way to grasp these fragments of knowledge and the theory of algorithmic information will provide the proper context for its definition.

2. Learning as Characterization of Structure

This chapter contains a short introduction to the theory of algorithmic information. We present the Kolmogorov complexity K , together with some computable approximations of it. It is shown to motivate the minimum description length principle (MDL), a formalization of Occam's preference for simple theories. We also present a distinction between a deterministic and a random component of an infinite object.

We concentrate then on an application of the above concepts: Solomonoff's theory of problem solving and learning as aspects of the same continuous process. We detail some difficulties and some consequences of it.

2.1 Induction Based on Algorithmic Probability

The ability to perceive the similarity of apparently different situations is essential to learning. Development of various measures for similarity is an old concern in pattern recognition and learning. Besides using such a measure in classifying situations, the important task is to find ways to extract that common essence that makes the two situations similar and determines a category. Memorizing that common essence is better than memorizing the situations themselves. On one hand, it is probably shorter. On the other hand the structure that was common to the two situations has a better chance at showing up later in a third one.

Similarity has been long recognized as an important theme in learning research. It is the basic concept in classification, pattern recognition, incremental concept formation, analogy. Most of the attempts, however, presumed a description language that makes easily accessible (computationally) those attributes that constitute the common part of objects or situations. Nevertheless, *the entire structure of those objects (that is, of their descriptions accessible to the learning algorithm) is potentially relevant for assessing similarity.*

Therefore it is natural to attempt to describe some canonical aspects of the structure of any object, that is, aspects that are, to a certain extent, independent of the description language one starts with. This is why we follow here a line of research that originated independently in the works of Solomonoff, Kolmogorov, Martin-Löf and Chaitin. All these link the description of an object to generator programs that output that description when run on a universal Turing machine U . While Solomonoff combines all generators of a given object, Kolmogorov and Chaitin concentrate only on the shortest generator. Chaitin and Levin subsequently showed that, if one considers only self-delimiting programs, that is programs that can be distinguished by U when they are concatenated on U 's tape without a separator symbol, the two approaches become essentially equivalent [Chaitin-78].

[Solomonoff-64] looks at the basic induction problem: the extrapolation of a given sequence, and arrives at the necessity to define an a priori probability on strings. In other words, given a string $s \in \{0,1\}^*$, we have to determine whether **0** or **1** is the most likely continuation for s . Let us say that an a priori measure $P: \{0,1\}^* \rightarrow [0,1]$ were known. Then we could base our prediction on the relative size of $P(s0)$ and $P(s1)$.

Assume that the given string s is the output of an unknown process the structure of which determines the probability of the next character. Such a process can be seen as an *explanation* for s . Solomonoff proposes a measure that takes into account all the possible explanations. Formally he considers s to be the output of a universal Turing machine U that starts with the *explanation* (or the process description) written on its tape. If e_i is an explanation, we can write $s=U(e_i)$.

Among all possible strings e_i from which U would produce s we would like to consider only those that are self-delimited, that is U will scan e only once and it will know where e ends. Therefore we cannot have one e that is a prefix of another. The explanations set E will be a prefix set. As a prefix set E satisfies the Kraft inequality

$$(a) \quad \sum_{e \in E} 2^{-|e|} < 1$$

Also, the explanations in E can be regarded as disjoint, so their contributions to the total probability of s can be added together. This allows the following definition of an algorithmic probability P_U of a string s :

Definition 2.1.1 $P_U(s) = \sum_{U(e)=s} 2^{-|e|}$

This definition also supports several intuitions:

- (i) The probability of s is the proportion of strings e that account for s (generate s given U).
- (ii) The higher probability of a string, the smaller the possible encoding of it.
Definition 2.1.1 can be seen as an inversion of Huffman coding: computing the probability from the code length.
- (iii) In a prefix set explanations can be considered as independent events.
- (iv) The simplest (shortest) explanation has the largest contribution to the probability (Occam's Razor).

2.2 Algorithmic Complexity and the Quantity of Information

[Chaitin-75,77,78] expands Solomonoff's concepts into a theory of algorithmic information. Given a universal Turing machine U and considering only self-delimiting programs he defines three basic concepts in reference to a string s . U has a program tape, a work tape and an output tape. Since the program is self-delimiting the program tape head will scan it once, stopping on the last character of the program.

Definition 2.2.1 $P(s)$ is the probability that U will produce s starting with any string on its program tape. These input strings e are assumed to be a priori distributed according to $2^{-|e|}$.

Definition 2.2.2 The entropy $H(s) = -\log P(s)$.

Definition 2.2.3 The algorithmic complexity, or information, $K(s)$ is the length of a minimal program that produces s .

For Definition 2.2.3 Chaitin uses $I(s)$, but, since this is the original Kolmogorov complexity, we chose to write it $K(s)$ throughout this text. $K(s)$ is also the largest contributing component in Solomonoff's probability measure, and, in fact, Chaitin shows that

$$(a) H(s) = K(s) + O(1)$$

therefore Definition 2.2.1 can be replaced by $P(s) = 2^{-K(s)}$.

Chaitin uses H for the algorithmic entropy throughout his papers. Again, since this is essentially equivalent to the Kolmogorov complexity, we shall use K instead of H . This also leaves us the possibility of using H to denote Shanon's entropy.

Definition 2.2.4 $K(s,t)$ is the entropy of both strings s and t , that is, of the string " s,t ". Chaitin calls it the joint entropy.

A similar joint version is defined for $P(s,t)$.

Definition 2.2.5 $K(s:t) = K(s) + K(t) - K(s,t)$ is the mutual entropy of s and t , that is, the extent that s and t are similar

Definition 2.2.6 $P(s|t) = P(s,t) / P(t)$ is a conditional probability, and $K(s|t)$ is defined as $-\log P(s|t)$.

Consequently

- (b) $K(s,t) = K(t) + K(s|t)$ Note that $K(s|t)$ is the minimal length of a program that has to be added to a generator of t in order to obtain a program that generates s .

Chaitin extends the above concepts from strings to RE sets of strings, such that the measures applied to singleton sets correspond to the definitions 2.2.1 ... 5. This enables him to state an information theoretic version of Gödel incompleteness theorem: *One cannot prove that an object s has a complexity $K(s) > n$ within a formal theory whose axioms have complexity $K(axioms) < n$.* Several properties of these concepts are established.

- (c) $K(s,t) = K(t,s) + O(1)$ There is essentially the same information in s,t as in t,s
- (d) $K(s) \leq K(s,t) + O(1)$ There is at least as much information in s,t as in s
- (e) $K(s,t) \leq K(s) + K(t) + O(1)$ One way to generate s,t is to concatenate a program for s and one for t , but there might be other ways, yielding shorter programs
- (f) If $|s| = n$, then $K(s) \leq K(n) + n + O(1)$ One possible self-delimiting program for s is made up of a self-delimiting program for n followed by the n characters of s itself

The constant addition $O(1)$ will not be further mentioned but will be implicitly assumed in all equations involving the Kolmogorov complexity.

Definition 2.2.7 An infinite string s is k -random iff $(\exists n_0)(\forall n > n_0)(K(s_n) > n - k)$ Therefore s is k -random if, beyond a certain length, none of its prefixes can be compressed more than k bits.

Chaitin uses the algorithmic entropy to define a degree of organization for geometrical patterns. It probably works for any metric space. This is the d -diameter complexity of an object X .

Definition 2.2.8 $K_d(X) = \min [K(\alpha) + \sum_i K(X_i)]$
where $\{ X_i \mid \text{diameter}(X_i) \leq d \}$ is a partition of X

It is defined as the minimum number of bits needed to describe X as an assemblage of parts, each with diameter smaller than d . All partitions $\{X_i\}$ of X into such objects are considered together with the description α of how should they be assembled. All pieces should have separate descriptions without cross-references. He suggests that a study of

$$(g) K_d(X) - K(X)$$

when d varies between 0 and the diameter of X would result in a kind of spectrogram of the object X . He uses this measure in several examples of natural spatial structures (a container of gas, a crystal, etc.). We shall refer it again in section 3.6 where we discuss *basic regularities* and *hierarchies*.

Two of the above concepts will be especially useful throughout this text. They are

$K(x : y)$ - the mutual algorithmic structure, or mutual entropy, and

$K(x | y)$ - the conditional complexity, or the additional algorithmic information necessary to express an object x once we have a program to produce y .

From definition 2.2.5 and (b) we can infer

$$(h) K(x,y) = K(x : y) + K(x | y) + K(y | x)$$

This decomposition leads us to a few tentative definitions for the common information of two objects. For two objects x and y the common information (relative to a universal Turing machine, like all definitions in this chapter) would be embodied by a program p_z that can contribute to the computation of both. That is, there are two other programs $p_{x|z}$ and $p_{y|z}$ such that $p_z p_{x|z}$ is a program for x and $p_z p_{y|z}$ is one for y . To make this definition work, we would like p_z to cover all common aspects of x and y , therefore $p_{x|z}$ and $p_{y|z}$ should not have any common part. We can force this algorithmic independence by asking $K(p_{x|z} : p_{y|z})$ to be very small.

[Chaitin-78] presents similar proposals for a measure of common information:

- (i) the maximum information of a string that can be extracted easily from both x and y
- (j) the minimum mutual information between the pair x,y and a string z relative to which x and y are nearly independent.

- (k) the minimum information of a string z relative to which x and y are nearly independent.

We shall use the following notation: x^* and y^* are minimal programs for x and y respectively, ϵ is a small tolerance, and the quantity $K(x:y | z)$ is defined as $K(x|z) + K(y|z) - K(\langle x, y \rangle | z)$. With this, the above definitions can be expressed as

$$(i) \max \{ K(z) \mid K(z | x^*) < \epsilon \ \& \ K(z | y^*) < \epsilon \}$$

$$(j) \min \{ K(\langle x, y \rangle : z) \mid K(x:y | z^*) < \epsilon \}$$

$$(k) \min \{ K(z) \mid K(x:y | z^*) < \epsilon \}$$

[Lloyd-90] contains an informal discussion of algorithmic complexity and illustrates its versatility. [Chaitin-78] and [Bennet-85,88] offer interesting examples of application of complexity based concepts to characterization of biological systems.

2.3 Structure and Randomness

We shall focus here on the *structure* of data. K is a useful measure for the *total amount of information* contained in an object. However, natural learning is concerned with *the order in data, that is, those fragments of information that are used repeatedly and can support prediction*.

Let us start with an example. Suppose that we are looking at a fir tree and we would like to draw an image of it for a children's book. A fir tree is a fairly complex object but we can probably get away with a simple drawing. Maybe a green triangle would do. Better, let us identify the number and position of the large branches, and sketch them symmetrically disposed around the trunk. If we want to be more precise, we should also sketch some thin lines around the branches to represent the needles.

Is this a description of the fir tree? What about the enormous quantity of information contained in the exact disposition of branches, large and small, the exact shape, dimensions and positions of needles, the scaly cones ... ? Not important - we could answer. The simple drawing is enough to identify a fir as a class, and, with some care for the details of large branches, even this particular individual. We are intuitively aware that we have picked a small but essential part from the total information.

We notice this phenomenon in attempts to describe a French text as a Markov process. Knowing, for each quadruple of letters, the probability $P(a_4 | a_1a_2a_3)$ of a_4 following the string $a_1a_2a_3$, is certainly less than knowing French. And yet, with a random number generator and these probabilities, we can generate a text that looks remarkably like French.

The measures surveyed so far assign the highest complexity to random objects. For a machine learning point of view there is really nothing, i.e. no regularity, to learn in a random object. What a learning agent is expected to discover in an apparently random universe are constraints that prove that the universe is not as random as it seemed, and that certain rational predictions about it are possible. On the other hand, there is (almost) nothing to learn from a trivially regular object either. *Here we follow a couple of attempts to separate order from randomness.*

[Cover-85] mentions another complexity measure K^* proposed by Kolmogorov. Let us denote by $K(x)$ his standard measure, that is the size of the shortest program that outputs the string x . Here are two new definitions:

Definition 2.3.1 Let $h(k, x) = \min \{ \log(|S|) \mid x \in S \ \& \ K(S) < k \}$

Definition 2.3.2 Let $K^*(x) = \min \{ k \mid k + h(k, x) = K(x) \}$

The first specifies the smallest set of complexity less than k that contains x . The second considers a two step description of x : a program of length k for the set from 2.3.1, and the ordinal (of length $\log(|S|)$) of x within that set.

$K^*(x)$ denotes the smallest k for which this two step description of x is as short as $K(x)$. K^* is a definition that separates the actual “structure” of x , (that is the regularities that allow a k bit description for S , or some generalizable property that can be predicated of x) from the “random” part of x , (that is a number that identifies x among the other strings that share its regularities). In virtue of the above definitions, the ordinal of x within S is truly random since any regularity in it could be moved into the description of S and exploited to obtain a smaller S .

We may call S the structure class of x , that is the set of all strings that have the same structure as x . A computable approximation of K^* may be a way to characterize an object (string) by its structure class and a random parameter that identifies the object within the class.

Such a distinction is made more precise by [Koppel-87] who continues the idea of separation between structure and randomness. He considers that an infinite string of observations of a physical phenomenon reflects both a deterministic structure and a random component.

From this point of view, a totally random infinite string $s_1 \in (a+b)^*$, and the infinite string $s_2 = aaaaa...$ are equally uninteresting since s_1 has no structure and s_2 has a trivial one. The algorithmic information of the first is infinite, the one of the second is almost null. However, intuitively, they both have simple explanations: s_1 is *a random string*, s_2 is *a string of a's*. They both lack sophistication.

In order to formalize this intuition consider an infinite sequence s whose structure is to be discovered. If a minimum size program P , and an infinite sequence d are found such that $P(d) = s$ and d is incompressible, then P contains the structure of s and d contains the random choices needed to instantiate P to s . Here are a few definitions and results from [Koppel-87].

For a fixed universal Turing machine U , we write $U(P,D) = S$ to mean that the program P reads the input sequence D and produces the sequence S . We also say that (P,D) is a description of S . The length n prefix of S will be denoted S_n . An empty program will be denoted by Λ . It has 0 length and it is considered self-delimited.

Definition 2.3.3 The Kolmogorov complexity of S ,

$$K(S) = \min \{ |P| + |D| \mid (P,D) \text{ is a description of } S \}$$

Definition 2.3.4 P is a c -minimal program for S and (P,D) is a c -minimal description of S iff

$$|P| + |D| < K(S) + c$$

Definition 2.3.5 c -sophistication(S) =

$$= \min \{ |P| \mid \text{where } (\exists D)(P,D) \text{ is a } c\text{-minimal description of } S \}$$

Definition 2.3.6 P is a weak-compression program for S iff

$$(\exists c)(\forall n)(\exists D_n)(P,D_n) \text{ is a } c\text{-minimal description of } S_n$$

Definition 2.3.7 A weak compression program is a *compression program* if

$$(\forall n)(D_{n-1} \text{ is a prefix of } D_n)$$

Definition 2.3.8 weak-sophistication(S) =

$$= \min \{ |P| \mid P \text{ is a weak compression program for } S \}.$$

Koppel shows that weak-sophistication(S) is defined for every S .

Definition 2.3.9 sophistication(S) =

$$= \min \{ |P| \mid P \text{ is a compression program for } S \}$$

A *transcendent* string is one with nonfinite sophistication, that is with

$$\lim_n c\text{-sophistication}(S_n) = \infty,$$

Let Λ be the empty program. With these definitions, a string α is random iff (Λ, α) is a minimal description for it, therefore its sophistication is 0. This entails the popular concept of randomness already expressed in definition 2.2.7.

Koppel shows that the definition of compression program is independent of the choice of U . Therefore the compression program does determine the structure of a string. To this purpose he defines the set $I_k(P,D,n)$ of all k -bit extensions of the prefix S_n compatible with a description, (P,D) , of S :

Definition 2.3.10 Let (P,D) be a description of an infinite string S .

$$I_k(P,D,n) = \{ s \mid |s| = n+k \ \& \ (\exists x)(s \text{ is a prefix of } U(P,D_n x)) \}$$

For two descriptions (P,D) and (P',D') , and two extension sets $X = I_k(P,D,n)$ and $Y = I_k(P',D',n)$, the extent to which $X=Y$ is the extent to which the two descriptions determine the same structure. He shows that $\lim_{n \rightarrow \infty} (\max \{ k \mid X=Y \}) = \infty$

Since all the above definitions are based on a partial recursive function U , a minimal description cannot be computed for every S . However Koppel gives an enumeration algorithm that finds, in the limit, the structure of a string S , provided that S is F -shallow, that is, the runtime its compression program needs to produce S_n is bound by $F(n)$ where F is a recursive function.

Based on the above concepts Koppel argues (against Popper's paradigm) that a theory, that is currently held valid, need not be refuted by new data in order to be replaced. A more complex theory may become a better explanation for the data, if it achieves higher compression.

We shall use the concepts developed by Koppel to make the following claims:

a) *A short but non minimal description (P,D) of a string still captures a fragment of its structure. A remainder of structure - that we shall call a residual - should be inducible from the process of building minimal descriptions (compressing) for P and D . (see chapter 3)*

b) *An agent executing a random walk through a well structured universe should be able to induce a minimal description of its infinite string of sensorial information. This description should separate the randomness introduced by the exploratory walk, from the fixed structure of that universe. (see chapter 4)*

2.4 The Minimum Description Length (MDL) Principle

Algorithmic probability can be used to translate the classic Bayesian approach to hypothesis selection. The resulting MDL principle formalizes the Occam's Razor intuition that the simplest explanation is the best. Unfortunately, the application of MDL requires computable approximations for the algorithmic complexity K . In this section we review the derivation of MDL and some previous approaches to approximating K . We also present a generalization to these approximations and some instances of the MDL principle that we shall use later.

2.4.1 Bayesian motivation

A Bayesian approach to explaining the data d by a hypothesis h would be to choose an h that has the highest probability given the data. That is to maximize $P(h|d) = P(d|h)P(h)/P(d)$. This is equivalent to minimizing $[-\log P(h|d)]$.

$$(a) \quad -\log P(h|d) = -\log P(d|h) - \log P(h) + \log P(d) \quad (\text{Bayes})$$

Since the last term in (a) is a constant, h should be chosen to minimize (b).

$$(b) \quad (-\log P(d|h)) + (-\log P(h))$$

Assume now that hypotheses are represented by strings. In this case we can express the a priori probability of a hypothesis as

$$(c) \quad P(h) = 2^{-K(h)} \quad \text{where } K(h) \text{ is length of a minimal prefix code for } h$$

Then, the expression to be minimized becomes

$$(d) \quad K(h) + K(d|h)$$

The first terms of (d) can be interpreted as the length of a minimal description of h . The second is, literally, the length of a minimal program that needs to be added to a program for h in order to produce d . It can be seen as a cost of describing d as an instance of h , but also as a measure for the error of approximating d by h . These interpretations entail the following induction principle:

The best model that explains the data d is one that minimizes the sum: *the description length of the model plus the additional information needed by the model in order to produce the data*. In other words, the best model is the one that provides a minimal length description (encoding) of data. It is an application of the Occam's Razor principle that settles the debate between simplicity and accuracy of the model.

We shall later refer to this derivation of MDL to support the following claims:

- (i) The choice of a class C_i to classify an element x should be made to minimize the description of x : $K(C_i) + K(x|C_i)$. This will be used in section 3.5.2 as a criterion for fragmentation.
- (ii) The best generalization from a single example x is its compression program p . This will be used in section 5.2.2 as a guide for explanation based generalization.

2.4.2 Approximations for Algorithmic Complexity

One of the obstacles to the systematic application of the MDL principle and the algorithmic complexity is the fact that K is not computable. Good approximations are needed. We are going to look here at three previous attempts. The algorithmic information theory allows us to present them in a uniform way. They are all based on replacing the universal Turing machine U by a more restrictive one V . We shall use $\text{plex}(s)$ to denote a computable approximation of $K(s)$.

[Lempel-76] restricted the programs producing s to the class of finite state machines. Given some representation for them, $\text{plex}(s)$ is defined as the shortest description length for a finite state machine that outputs s . It provided the basis for their text compression algorithm.

[Rissanen-86] is concerned with computing complexity of integers, since, to a suitable universal machine all integers are codes for partial recursive functions. Let N be an integer and the string s its binary representation. Rissanen's complexity is based on a strategy for making s , self-delimiting. This is how it works. Resuming the comment of 2.2(f) we can construct a self-delimiting program for s by concatenating the string itself at the end of a self-delimiting program for n , the length of s . Carrying this further, what is a program for n ? One answer is: a string representing n in base 2, concatenated at the end of a self-delimiting program for $\log n$. Rissanen used this recurrent argument to propose the following approximation for $K(N)$.

- (a) $\text{plex}(N) = \log^*(N) + c$ where
- (b) $\log^*(N) = \log(N) + \log(\log(N)) + \log(\log(\log(N))) + \dots$
- (c) $c=2.865$ (such that $\sum_{n=1, \infty} 2^{-\text{plex}(n)} = 1$)

[Hart-87] shows several applications of the MDL principle to the estimation of structure. Hart gives a language theoretic approach to approximate model complexity.

will have the same complexity since they involve the same *amount* of decision. However e_1 and e_2 each use the same production 7 times, hence they are 'simpler' than e_3 by an absolute standard. In fact each of them can be generated by a simpler grammar.



A way to capture these second degree regularities would be to combine Hart's measure with a similar one applied on the program p itself. This would be in the spirit of the measure proposed by Rissanen for integers. The task is to define a grammar that covers both levels: the data and the program, that is, the derivation tree.

In conclusion, we shall use $plex(x)$ instead of $K(x)$, and accept the restricted class of computations, introduced by it, as an implicit bias.

2.5 Unification of learning and problem solving

To end this presentation of the algorithmic information theory, let us present a possible application, then try to discuss it. [Solomonoff-86,90] attempts to use algorithmic probability as an all-encompassing approach to learning and problem solving. His method is to encode all relevant knowledge in probability distributions. He shows that they can both guide problem solving, and encode the acquired experience. He considers two very general problems and shows that many specific problems of AI research are instances of those. Here they are (Σ_1 Σ_2 are alphabets, R is the set of real numbers):

- (i) Inversion. Given $M: \Sigma_1^* \rightarrow \Sigma_2^*$ and $s \in \Sigma_2^*$, find, in minimum time, an $x \in \Sigma_1^*$ such that $M(x)=s$
- (ii) Time limited optimization. Given $M: \Sigma_1^* \rightarrow R$, find, within a time limit T , an $x \in \Sigma_1^*$ such that $M(x)$ is maximum

Note the generality of the above formulations. For example, the problem of finding a proof P for a theorem T given the axioms A can be cast as an instance of (i). Assume that we have a program M that checks whether P is a proof for T . Then $M(A,T,P)$ is a boolean function and proving T is equivalent to finding a string P such that $M(A,T,P)=\text{true}$. Also, many problems that can be formulated as a search for a best solution can be cast as instances of (ii). For example finding a function from a given class that best fits a set of examples is an instance of (ii).

Solomonoff uses a generate and test approach based on an enumerative procedure from [Levin-73]. This requires, for all strings $x \in \Sigma_1^*$, the knowledge of two quantities:

- $p(x)$ - the probability that candidate x be a solution to the problem
- $t(x)$ - the time (number of computation steps) necessary to test the candidate x .

The use of the above knowledge can be seen in terms of a game. We are given an infinite series of bets that are to be tried sequentially in order to win some constant prize. For each bet x , $p(x)$ is the probability of x to be the winning bet, and $t(x)$ is the cost of betting on x . A bet that fails cannot be tried again. All we have to do is to keep betting until one bet succeeds. In order to minimize the cost of winning, one has to try the bets in the ascending order of $t(x)/p(x)$.

Note that, without any previous experiments, the time values $t(x)$ cannot be known. However algorithm 2.5.1 is a good approximation of this betting procedure.

Algorithm 2.5.1

Set a small time limit T .

Loop

For all candidates x with $1/p(x) < T$ do

Test x by running it at most $p(x)T$ steps (to ensure that $t(x)/p(x) < T$)

If a solution has been found, then stop

Set $T \leftarrow 2T$ and continue the loop

Solomonoff shows that if there is a fastest solution x_0 , and all information necessary to solve the problem is expressed in the probability distribution p , then the above method will take at most $2t(x_0)/p(x_0)$ to solve the problem.

Obviously, a problem solver with no experience has no knowledge of $p(x)$ either. The probability can first be approximated as $p(x) = |\Sigma_1|^{-|x|}$. Once we have a problem-solution pair PS , we have to include this knowledge into the probability distribution (d). Here Solomonoff is sketchy. He says that the ensemble (d, PS) has to be compressed and newer, shorter codes should be derived for the operators x . The new distribution will be derived from the length of those new codes. He also notes that compression is a time limited optimization problem, therefore the same principles can be applied to it. He gives some simple examples that do not seem to illuminate the general picture. In the remainder of this section we present our perspective on this subject.



Investigations. We proceed now a few steps further, and try to make some details a little more explicit than they are presented in the papers referenced above. Meanwhile, we shall point to some difficulties involved, and we shall also draw some interesting conclusions.

First, let us take a close look at inversion problems. Let us say that the solutions we are looking for are strings over an alphabet A . Following Solomonoff we shall call these strings operators. Given a problem p , and the recursive predicate $Solution$, we have to find an operator $s \in A^*$ such that $Solution(s, p)$.

The algorithm 2.5.1 presumes a probability distribution on operators. Let us make some assumptions on the source of that distribution. Assume that we have some experience at this type of problems, having solved some of them in the past. That experience embodies the knowledge of previous solution-problem pairs:

$$(1) \text{Exp} = \{ [s_i, p_j] \mid \text{Solution}(s_i, p_j) \}$$

Then, our problem-solving experience may be expressed as a function E that takes a body of experience and a new problem and returns a probability distribution on operators, an educated guess on their chances to solve the new problem:

$$(2) E(\text{Exp}, p_{\text{new}}, s_i) = P(s_i)$$

Learning is constructing a program for E . How is this possible? Let us first clarify what is $P(s_i)$. We have to assume a representation for Exp where each past problem and its solution are expressed as strings. Although we defined Exp as a set, the temporal order of the problems may matter. (For example, teachers frequently follow a problem by a related one.) So let Exp be represented by a string description $d(\text{Exp})$ that consists of a concatenation of descriptions of past problem-solution pairs:

$$(3) d(\text{Exp}) = d(p_1) s_1 \dots d(p_n) s_n \quad \text{where indices reflect the temporal order}$$

In order to simplify the notation we shall assume that p stands for $d(p)$, a string description of the problem p , and we shall rewrite $d(\text{Exp})$ as $p_1 s_1 \dots p_n s_n$

Then, proposing an educated guess for the solution to a new problem p_{new} is reduced to the following induction problem: Which $s \in A^*$ is the best continuation for the string $d(\text{Exp}) p_{\text{new}}$? Within the algorithmic complexity framework, and similar to the one in section 2.1, an answer to this question should be based on the probability

$$(4) P(s \mid d(\text{Exp}) p_{\text{new}}) = P(d(\text{Exp}) p_{\text{new}} s) / P(d(\text{Exp}) p_{\text{new}}) \quad \text{or}$$

$$(5) P(s \mid d(\text{Exp}) p_{\text{new}}) = 2^{-K(d(\text{Exp}) p_{\text{new}} s)} + K(d(\text{Exp}) p_{\text{new}})$$

But this is precisely what E should compute if K were recursive. Hence, a program for E will be one able to compress the following strings

$$d_1 = p_1 s_1 \dots p_n s_n p_{\text{new}}$$

$$d_2 = p_1 s_1 \dots p_n s_n p_{\text{new}} s$$

If we assume now that $\text{plex}(x)$ is an approximation for $K(x)$ that reflects our current compression skills, then, a program for E will be

$$(6) E(\text{Exp}, p_{\text{new}}, s) = 2^{\text{plex}(d_1) - \text{plex}(d_2)}$$

Discussion Some of the difficulties of this characterization spring from comparing the resulting distribution to the one prescribed by intuition in some special cases:

a) How good will this guess be when s is the actual solution of p_{new} ? Let us consider the ideal case where we have a perfect method to generate solutions from problems. This is equivalent to having a function f with the property

$$(\forall p) \text{ Solution}(f(p), p)$$

Having f^* , a program for f , and using the prefix code property of descriptions, allows us to construct a simple program g^* , such that

$$\begin{aligned} g^*(f^* p_1 \dots p_n p_{\text{new}}) &= d_1 = p_1 f(p_1) \dots p_n f(p_n) p_{\text{new}} \\ g^*(f^* p_1 \dots p_n p_{\text{new}} 1) &= d_2 = p_1 f(p_1) \dots p_n f(p_n) p_{\text{new}} f(p_{\text{new}}) \end{aligned}$$

$$\text{Let } d = g^* f^* p_1 \dots p_n p_{\text{new}}$$

$$K(d 1) - K(d) = K(d) + K(1|d) - K(d) = K(1|d) \leq K(1) = 1 \iff P(s) = 2^{-K(d)} - K(d 1) \geq 2^{-1}$$

Remember, however, that the above inequalities involve an implicit constant term. Still, we conclude that a perfect solution will be presented as a highly probable guess.

b) The predicate `Solution` should be accessible to algorithm 2.5.1 only as an oracle. If we assume the contrary, we can easily build the program f^* mentioned above out of the program for `Solution` and a simple enumeration. This implies that we have a fairly short program that, although long to run, can compute s out of p_{new} , regardless of any experience.

c) If s was the solution to a past problem, it will receive a fairly high probability, regardless of its relevance to p_{new} . This is because, using an experience of n problems, s can be identified and copied at the end of the string d_1 by a program that incorporates n , and thus, of length $\log(n) + O(1)$.

d) The evaluation of $\text{plex}(d_1)$ suggests that each new problem should cause a complete recompression (therefore a reorganization) of the entire previous experience. In a practical setting, this is unlikely. If the experience is large, only its structure (see section 2.3) is likely to be kept and the random part discarded. This will make complete recompression impossible.

e) One possible compression of experience may take place through discovery and construction of common problem features and common operator features. Common features will define problem and operator types. Statistical correlation of problem and operator types will allow compressing experience into rules like:

If p_{new} is of problem-type A, and s is of operator-type B, then assign to $P(s)$ a value based on $P(B|A)$.

In this case, the simplest accumulation of experience consists in reevaluation of the correlations. A better one will involve construction of new features, therefore of new types.



Now let us concentrate on how the new solution becomes experience. We propose an algorithm based on repeating indefinitely the following two steps:

Algorithm 2.5.2

Loop

(1. Solve)

Let $\text{problem}_{\text{new}}$ be the new training problem.

Use $E(\text{problem}_{\text{new}})$ to enumerate operators and algorithm 2.5.1 to solve it.

Let operator_i be the solution thus obtained.

(2. Learn)

Let $\text{compression_problem} = \text{"description}(E), \text{problem}_{\text{new}}, \text{operator}_i\text{"}$

Use $E(\text{compression_problem})$ to enumerate operators and algorithm 2.5.1 to solve it.

Let operator_k be the solution thus obtained.

Apply operator_k to $\text{compression_problem}$ to obtain the new definition of E

To the above algorithm we would like to add the following remarks.

(1) **Learning learning.** Both parts of the loop contain problem solving activities, but only the first one is incorporated into E . For the compression ability to evolve, the solution to the (2.Learn) problem should be learned in its turn. The basic strategy, $\text{Learn}(\text{Solve})$, would change to $\text{Learn}(\text{Learn}(\text{Solve}))$, $\text{Learn}(\text{Learn}(\text{Learn}(\text{Solve})))$, etc.

(2) **Unsupervised learning.** The learner starts with an initial set of operators powerful enough to solve both the external problems and the internal compression problems. Sequences of operators that are assigned new, higher probabilities - i.e. are compressed into shorter codes - will become new operators. But this creation of new concepts depends on a careful training that presents simple problems first, to prompt the learning of simple new operators and continues to progressively build on those. This way, the time, necessary for one run through the loop above, will be approximately constant. However, in a more natural setting where there is no trainer, the program can be changed to select problems at random and attempt to solve them for a predetermined time interval (patience, attention span ?). A problem that times out is dropped. With a fair distribution of problems, the hierarchy of concepts will should develop.

(3) **Cooperation of learning and solving.** In Algorithm 2.5.2 the function E has a minimal interaction with the solver (Alg. 2.5.1). E produces the next operator sequence, the solver tries it and responds with "yes" or "no". Assume that the solver is using the operator string as a sequence of search steps. With no experience, all operators will be more or less equiprobable and shorter sequences will be tried before longer ones. Later, though, some successful longer sequences should acquire higher probabilities through compression. Therefore the searches should evolve from breadth first towards best first.

Still, while one problem is tried, no learning occurs, since there is no solution yet. This can be changed by mixing the two steps Solve and Learn. One thing to be learned this way is, for example, that no operator sequence will succeed if a prefix of it failed before. The function E will be able to compute the chances of operators depending on a particular stage of the search instead of probabilities dependent on the entire problem.

(4) **Physics and mathematics.** We can imagine ever more elaborate ways of constructing operators, from operator schemata to type theories. Assigning the right codes to new operators will depend on studying their general properties. For once, many of the generated operators may be equivalent. A theory able to entail such equivalences would contribute to both Solve and Learn. Building such a theory would require study of operator properties independent of particular instances of operator application, like equivalence and subsumption, and would resemble mathematics.

A number of researchers [Lenat-78,83,84], [Haase-87,88], [Shen-90] attempted to automated this type of investigation. From their approaches resulted two requirements of discovery systems: (a) a criterion for focussing on new concepts and (b) a criterion for evaluating them. The context outlined above naturally supplies both. (a) Operators that solve problems are candidate objects for study. (b) Useful concepts will be those that are assigned short codes (high probabilities) since their impact on compression is explained by their large applicability. To conclude, we can distinguish between two aspects of learning: *physics* - learning domain dependent regularities -, and *math-*

ematics - learning general properties of the new description language elements that allow manipulation of descriptions. We shall motivate this in the context of constructive induction, in section 5.3.



In conclusion, a theory to explain learning in the context of problem solving at this level of generality and elegance, should be extremely attractive. It may hold the promise to unify many aspects of these activities, that were studied separately. It may offer important insights into any computational model of accumulation of experience.

3. Creation of Elements For A Private Language

This chapter will focus on induction problems and algorithms based on the MDL principle and will consider only algorithms that passively accept an infinite string from an unknown generator.

We formalize here the concepts of regularity, partial models, and complete models, all centered around the concern for compressed encoding. We show that a measure for compression offers good guidance against over generalization in induction from positive examples. We discuss model refinement and convergence.

Then, we introduce Finite Memory Models, finite automata with a vector representation for states. Although they do not have more recognizing power than finite automata, they can express more regularities. They are incrementally inducible with the algorithms mentioned above, and will prove a useful class for the next chapter.

Finally, we define Libraries as collections of partial models. We discuss induction of Libraries and its relation to clustering algorithms. We introduce then the concept of basic regularity as building block for hierarchical structures. We show how basic regularities can be identified by local maxima of compressibility as a function of model complexity. This allow us to define stepwise induction as a general strategy for avoiding the intractability of certain induction problems.



So far, *we have looked at algorithmic information theory as an unbiased way to describe the structure present in data.* This is what a completely passive learner can hope to perceive, without a priori information, in the data presented to it. It has been argued [Fodor-75] that an innate *language of thought* must be postulated for any intelligent agent in order to make possible the acquisition of natural language. This sparked a long debate about what cognitive abilities should be postulated before any learning becomes possible.

We shall resume this theme at the beginning of the next chapter. In this one we start looking at an abstract, computational model of learner - an algorithm that only reads an infinite string of data, but we continue, in the next chapter with more natural models of learning agents that can perceive and act on a well defined environment. In the process, we sketch how a private language vocabulary might emerge.

One could claim that learning is already a quality of any evolutionary scenario of life development, since certain stable features of the environment are selecting certain individuals from a population. Assume that every new generation introduces some random variations from the previous one. A selection means eliminating some of this ini-

tial variety. What does this mean in the framework surveyed in the previous chapter? Only individuals adapted to some environmental constraints survive. Therefore, as a result of selection, some of the environmental information is shared by the surviving population. In other words, *selection entails a loss of algorithmic information*.

This seems to contradict the intuition that the complexity of life-forms increases during evolution. As they reflect more of the stable environmental constraints, the life architectures become less random and their algorithmic complexity should indeed decrease. On the other hand, their amount of structure, that is, their sophistication increases, and this is perceived as growing organization.

At an individual level, one could equally claim that classical conditioning is a simple prototype of learning, since it associates a specific response to a specific stimulus. However, unless the presentations of the stimulus are always identical, a theory of learning has to explain how does an agent develop the ability to identify the presence of a certain stimulus in a large variety of situations. Two standard ways to attacked this problem are:

- by making the stimulus explicit in the presentation of data, that is by providing the learning algorithm with a recursive predicate to detect the presence of stimulus;
- by providing a training sequence (data items marked '+' or '-' for the presence or absence of the stimulus respectively) and a class functions computable from the data items guaranteed to contain a stimulus detection predicate.

In the absence of either detection predicate, or training sequence, a learning algorithm can still search for something in data: its intrinsic regularities, i.e. its structure. This approach is perfectly compatible with assisted learning too, since one of the regularities of a training sequence is the stable correlation between the presence/absence of stimulus and the accompanying '+' or '-' marks. The research in induction, in analogy and in unsupervised concept formation is relevant here. Since the progress of learning cannot be checked against a training sequence, the main criterion guiding that research is the ability to predict features of future data.

An interesting example, from our point of view of discovering intrinsic structure in data, is presented in [Gentner-83]. In order to find an unbiased way to determine a possible analogy between two logical theories, the author deliberately ignores the interpretation of different predicates, and considers, from a purely syntactical perspective, the graph constructed from terms and binary and higher arity predicates. An isomorphism identified between such graphs induces an analogy between the domains of the corresponding theories.

Certainly, unsupervised learning can be seen as induction of a partially recursive function that can generate the infinite sequence of data, after seeing a long enough prefix of it. Presented this way the problem is, at best, intractable. In this chapter we propose a model of learning in which regularities discovered in data are accumulated and possibly revised in a continuous process. The result will be a dictionary of basic concepts, necessary constituent terms for any theory that attempts to account for the entire sequence.

The language in which learning takes place, that is the set of formulae that constitute the search space of the learning algorithm has been long recognized as a potentially limiting bias. If it is too restrictive, it may not be able to express certain concepts. If it is too rich - for example the entire set of recursively enumerable functions - induction becomes intractable. [Utgoff-84] showed that one can start with a small bias and presented several methods to add elements to the language when they become necessary.

We are obviously dealing with the same problem of finding or constructing the appropriate bias - or language - to express the regularities in data. However, we shall investigate the following approach:

- Start with a very rich bias.

- Identify a basic conceptual level able to capture the simplest regularities in data.

- Use those concepts as primitive elements to construct higher level concepts.

We shall also show that a first learning stage can take place in a simple language, but the results of this stage can be further combined through composition and recursion in order to express more complex concepts.

3.1 Representations and Regularities

Induction problems are usually concerned with strings of symbols. Yet the knowledge acquired is not only about strings. Let us assume that learning of any object proceeds via a string representation of it. This entails the existence of a domain D containing the objects of study and a set of expressions (descriptions) that are strings over an alphabet Σ . They are linked through the representation mapping

$$(1) \quad r : \Sigma^* \rightarrow D.$$

We expect a description to contain all relevant information that the learning algorithm is supposed to discover. Precisely, for every relevant predicate P over objects in D , there must be a predicate Q over strings of Σ^* such that

$$(2) \quad (\forall s \in \Sigma^*) (Q(s) \Leftrightarrow P(r(s))).$$

Q represents P and is the regularity that has to be learned.

The learning algorithm may be given some a priori knowledge about the representation. For example it may *know* that the infinite presentation of data it is looking at consists of sets of tuples. This is in fact equivalent with endowing the algorithm with knowledge of the syntax for sets of tuples representation. The data presentation may look like this

$$(3) \quad \{[b,b,b] [b,a,a,a]\} \{[b,b] [b,a,a] [b,a,b] [b,b,a]\} \{[b,a,b,b] [b,b,a,a] [b,b,a,b]\} \dots$$

The given syntax will allow the algorithm to focus on the objects represented by it - the set of tuples - and, after parsing, to ignore the constructor characters $\{, \}, [,],$ and $,$. Without any a priori knowledge the algorithm will be bound to work at the level of strings of Σ^* , attempting to discover relationships between substring like $\{[a$ and $,a,b]$. In this case, one of the regularities to be discovered is the syntax of the sets of tuples representation.

However, once a certain syntactic regularity has been detected, the representation layer induced by it can be stripped, and learning may continue on the objects represented by it. They, in turn, may be the constituents of another representation. For example, in (3) the sets of tuples could be sets of consecutive binary numbers where **a** stands for **0** and **b** stands for **1**. (More about it in section 3.6 where this procedure is called step-wise induction.)

We imagine the learning algorithm as being passively fed a (potentially infinite) string produced by an unknown generator G . We call such an algorithm *passive* because it can only request the next symbol from the generator. Later we shall also con-

sider *active* algorithms that can *act* on G , affecting its behavior. Since the algorithm has access only to the strings in Σ^* but not to objects in D , we conclude that the *meaning* attached the strings by the mapping r will elude it forever. An active algorithm, though, can simulate the acquisition of meaning by relating its actions to regularities perceived in G 's behavior. (More about it in chapter 4)

The approaches to learning that we shall talk about are not confined to presentations of strings. They depend rather on the existence of compositional structures and the ability to define and analyze substructures. [Van Gelder-90] makes the point that representations need not be concatenative in order to be compositional. Representations obtained through concatenation have a special property: The expression for a composite structure physically contains the expressions of its constituents. For example, in (3) above, the string for the set of tuples $\{ [b,b,b] [b,a,a,a] \}$ contains the expressions of the two tuples as substrings.

Contrast this with Gödel numbers. They are an example of compositional representation that is not concatenative. Compositionality only requires a way to aggregate two expressions, and a way to uniquely retrieve them from the result. Neural network architectures have been criticized for not supporting concatenative representations. However, the above paper presents a few that support compositional representations. Therefore, we perceive the concepts developed here as applicable to the connectionist paradigm as well.

In conclusion, although we discuss regularities of strings, we do not restrict the generality of our theory.

3.2 Models and Codes

We shall give now a precise definition for the idea of regularity. Without loss of generality, we assume that all objects that we deal with accept descriptions that can be reduced to strings. For two strings s, s' we shall write $s \subset s'$ when s is a prefix of s' . Let C and S be sets of strings. The strings in S are descriptions of objects. The strings in C will be called codes. We adopt here the concept of process defined in [Koppel-89].

Definition 3.2.1 A process is a function $m : C \rightarrow S$ with the following property:

$$(1) \quad c1 \subset c2 \Rightarrow m(c1) \subset m(c2)$$

Processes provide plausible explanations for strings. Assume that we found a function m and a code string c to generate the prefix s_n , of length n , of an infinite string s . That is $m(c) = s_n$. This generative explanation for s_n should be consistent with the next character of s or the next larger prefix s_{n+1} , that is, we should be able to see s_n and s_{n+1} as generated by the same process. Therefore we should only have to add a fragment c' to the code c in order to generate s_{n+1} : $m(cc') = s_{n+1}$. This is why the function m has to be a process.

We only consider processes from a well specified class M . This will enable us to associate, to every process m , a positive complexity measure $\text{plex}(m)$ in order to approximate the algorithmic complexity $K(m)$. In some cases we shall relax $\text{plex}(m)$ to be simply the length of a description of m according to a given syntax. For example for a finite automaton m , we shall take $\text{plex}(m)$ to be the number of edges in the transition graph.

Definition 3.2.2 A model for a string s is a process that is able to compress some fragments $\{w_i\}$ of the string s , into the code strings $\{c_j\}$. That is, a process satisfying the following conditions

$$(2) \quad m(c_j) = w_i$$

$$(3) \quad \sum |w_i| > \text{plex}(m) + \sum |c_j|$$

We shall say that a model captures a *regularity* of the string s .

Definition 3.2.3 $\text{parses}(m, w) \Leftrightarrow \exists c \in C$ such that $m(c) = s$ and $w \subset s$

Definition 3.2.4 $\text{code}(m, w) = \min \{ |c| \mid w \subset m(c) \}$

For a prefix w parsed by m we measure the compression performance by the gain and the compression factor:

Definition 3.2.5 $\text{gain}_w(m) = |w| - (|c| + \text{plex}(m))$

Definition 3.2.6 $\text{press}_w(m) = \text{gain}_w(m) / |w|$

Definition 3.2.7 A *complete model* for a string is one that parses the entire string. A *partial model* is one that accounts only for some fragments of the string. (For example, the rule *aab is always followed by bbab or bbb* is a partial model)

Definition 3.2.8 A complete model that achieves the maximum compression represents a *complete regularity*. It is a compression program and, consequently, it reduces the data s to a random code string c . All other models represent *partial regularities*.

The above definitions are quite clear for finite strings. For an infinite string we can have only candidate complete models that parsed the prefix seen so far. They may eventually turn out to be partial models. We can extend the definition of gain and press to partial models m of a finite string s .

Example 3.2.1. Let s be a string in $\{0,1\}^*$ where the i^{th} position is 1 if i is prime and 0 otherwise. Here are three models for it:

m_1 enumerates the prime numbers. It can generate the entire string. It is a complete model, but also a complete regularity since it captures the entire structure of s . There is no random component to s , thus m_1 reduces s to a null code string.

m_2 produces a 0 in position i , if i is even, and uses a code string in $\{0,1\}^*$ to determine what to produce in odd positions. This is a complete model that captures only a partial regularity.

m_3 only predicts that every 1 in s is followed by a 0, except for the first 2 positions. This is a partial model. It cannot parse the entire string but only the 10 sequences.



A partial model can always be extended to a complete one by the addition of the identity function. Let m be a partial model. We can associate to m a complete model m' that compresses only those substrings of s that m can parse and leaves the rest unchanged. Let a parsing of the string s by the model m be defined by

Definition 3.2.9 $\text{parsing}(s,m) =$
a disjoint set $\{ w \mid w \text{ substrings of } s \ \& \ \text{parses}(m,w) \}$

For example if $s = u_1w_1u_2w_2u_3$, and $\{w_1,w_2\}$ is a $\text{parsing}(s,m)$, then the code for this parsing will be $u_1c_1u_2c_2u_3$ where $c_i = \text{code}(m,w_i)$. Now we can define a gain and compression factor for m on s by considering the best parsing:

Definition 3.2.10 $\text{press}_s(m) =$
 $(\max \{ \sum_{w \in p} (|w| - |\text{code}(m,w)|) \mid p \text{ is a parsing}(s,m) \} - \text{plex}(m)) / |s|$

Note that, in general, this measure cannot be computed incrementally, since there may be many overlapping parsings of s by a model m . However, a suboptimal incremental algorithm is possible if we have a parsing procedure m^{-1} that computes one of the inverses of m .

Procedure m^{-1}

Given a prefix s_n

we can enumerate code strings in order to find those $C = \{c_i\}$ such that $m(c_i) = s_n$.

Once those are found, we use the fact that m is a process, and we repeat the following:

Read the next character a of s .

For every c_i in C

Search all continuations $\{c_{ij}\}$ of c_i such that $m(c_i c_{ij}) = s_n a$.

The procedure m^{-1} can be started on any string and, unless m is a complete model for the string, it will fail after parsing some finite prefix w . We shall call this prefix the longest parsed prefix. Note however that this prefix is the longest only with respect to the particular inverse that m^{-1} computes. The algorithm 3.2.1 outlines the behavior of m^{-1} : it simply uses m to parse as much as it can, then it starts it again on the remaining characters.

Algorithm 3.2.1

```
Code <- empty string
While |s| > 0 do
  Let w be the longest parsed prefix of s and c = code(m,w)
  Append c to Code
  Remove the prefix w from s
  While m cannot parse s
    Remove the first symbol from s and append it to Code
```

There is a substantial amount of literature concerned with induction of functions able to enumerate a sequence. [Gold-78] defined the identification in the limit criterion. [Blum-75] gave an enumeration algorithm for recursively enumerable functions. They were both concerned with obtaining the 'simplest' function that generates the string so they considered candidates in increasing order of complexity.

However, we are interested in obtaining models that achieve the highest compressibility, since they embody the string's structure. Using Koppel's terminology, we would like to find a compression program for the string. Therefore we are not interested only in the first program that can generate the string, but we would like to enumerate all generators in order to assess their compression factor. It is easy to see that, for an infinite string, we may not be able to converge to a compression program. What is perceived as a regularity in a finite prefix may eventually be invalidated and the string may turn out to be random.

To resume, a standard enumerative approach has these two problems:

(a) In a fixed order enumeration of hypotheses a function is rejected as soon as it fails to parse a prefix of the string. The information encoded by that function's initial success is lost. In order to deal with this we shall consider the following solutions:

(i) **Induction by refinement.** We start with a small set of simple models as hypotheses and refine them when they fail. Let m be a candidate model parsing a string s . Let s_k be the prefix of s of length k . Assume that m failed at the k^{th} symbol of s , that is parsed s_{k-1} but not s_k . So m is a complete model for s_1, s_2, \dots, s_{k-1} but not for s_k . One can see m as a generalization of the first $k-1$ prefixes. We are looking for a refinement m' of m to be the simplest model more complex than m that is complete for the first k prefixes. Therefore m' should be a minimal generalization of m and s_k .

Without additional assumptions about the string s , that is about the way the data is presented, we are in a situation known as *induction from positive examples*. Section 3.3 investigates the general requirements for this approach and section 3.4 presents a class of models for which an incremental induction by refinement is possible.

(ii) **Induction of submodels.** We can relax the goal of inducing a complete model to the more modest one of accumulating a number of models, preferably small, each of which is able to parse some fragments of the string and to compress them significantly. Eventually, a complete model could be constructed by using them as subroutines. To put this in a better perspective, an ideal set of models would capture all the regularities of the string, that is its structure, while their arguments and order of invocation would embody the purely random component of the string. Submodels will be obvious candidate elements for a descriptive language. We shall investigate this in section 3.5.

(b) While the current hypothesis is still successful in parsing longer and longer prefixes, we should consider other hypotheses in hope for a larger compression factor. This makes the usual algorithms intractable not only in terms of time but also in terms of space. A solution to this is provided by:

(iii) **Stepwise induction.** Some complex structures are composed of simple ones. For such structures, induction of important building blocks should precede and simplify the induction of complete model. In section 3.6 we shall investigate criteria to identify building blocks, and the ways knowledge of building blocks can simplify further induction.

3.3 Induction by Refinement

A comprehensive framework for induction from positive and negative examples is the version space approach [Mitchell-78]. A simple and elegant theory of generalization and specialization as operators can be found in [Laird-87]. He calls them upward and downward refinements, respectively. He takes the concept of refinement operator that [Shapiro-81] used for first order logic and derives necessary and sufficient properties that enables it to work in any domain. We shall develop our theory in the same style.

Definition 3.3.1 $L(m) = \{ w \mid \text{parses}(m, w) \}$

The language of a model is its range.

Definition 3.3.2 $m \equiv m' \iff L(m) = L(m')$

Models with the same language are equivalent.

Definition 3.3.3 $m \ll m' \iff L(m) \subseteq L(m')$

Model m is less general than m' .

Assume that we have a generalizing operator ρ that, when applied to a model, generates a set of more general models.

Definition 3.3.4 ρ is an upward refinement [Laird-87] if $\rho(m)$ is a set of models with the property:

$$m' \in \rho(m) \implies m \ll m'$$

We shall also extend the action of ρ to sets of models: $\rho(A) = \bigcup_{m \in A} \rho(m)$

We want generalize a hypothesis only to correct a specific failure: that of failing to parse a symbol 'a' after having parsed a string w . Some generalizations in $\rho(m)$ may not fix it. To this purpose we would like to define a specific refinement set that contains common generalizations of m and wa :

Definition 3.3.5 $\text{refinement}(m, wa) = \rho(m) \cap \{ m' \mid \text{parses}(m', wa) \}$

Finite state machines and finite memory machines (presented in the next section) are examples of classes for which $\text{refinement}(m, wa)$ is not a vacuous concept.

Let us first present on a nonincremental algorithm, that is one that remembers the longest prefix of s seen so far and can test every new hypothesis on it.

Algorithm 3.3.1

```
1  k <- 1
2  H <- M0
3  Loop
4    Let sk = sk-1a
5    For all h ∈ H that cannot parse sk
6      Remove h from H
7      Add refinement(h,sk) to H
8    Output h that maximizes press(h) on sk
9    k <- k+1
```

Note that the algorithm is almost a breadth first exploration of the graph of the relation $<<$. The sets of hypotheses $H_0=M_0$, $H_1=\rho(M_0)$, $H_2=\rho(\rho(M_0))$, ... seem to be investigated in turn.

For this algorithm to work we have to assume that the following conditions are fulfilled:

(i) The predicate $\text{parses}(m,w)$ is recursive

This is needed by line 5.

(ii) $(\forall m \in M)(\forall a \in \Sigma)(\forall w \in \Sigma^*) (\text{parses}(m,w) \Rightarrow$
 $\Rightarrow \text{the function } \text{refinement}(m,wa) \text{ is recursive and its result is a finite set})$

This is needed by line 7.

(iii) There is a finite lower bounding set $M_0 \subset M$ such that

$(\forall m \in M)(\exists m_0 \in M_0)(m_0 << m)$

(iv) ρ is complete for M , that is $(\forall m \in M)$ there is a finite k such that $m \in \rho^k(M_0)$

These two conditions assure that every model in M is reachable in a finite number of refinement steps. However, since a hypothesis is refined only when it fails to parse, the exploration of the space of hypotheses doesn't proceed so orderly. So, it is possible that a certain model is never reached if, on every refinement path from M_0 to it there is a complete model of s .

This is an instance of an optimization problem, complicated by the fact that the

model that attains the optimum (maximum compressibility) may have to change with any newly read symbol of s . Suboptimal methods may apply. Hart, for example used a genetic search for finite automata models.

In order to turn the above algorithm into an incremental one, it is sufficient to add the following two conditions:

- (v) The new hypotheses must start parsing without having them to process the prefix seen so far. They have to be put in the state where they would have been after parsing that prefix.
- (vi) The code length produced while parsing that prefix must be computed in order to evaluate their compression factor

So let us say that there is a best model (maximum compressibility) for s in the class M , and eventually it will be added to H . Will it be output by the algorithm?

First we have to remark that once a hypothesis is a complete model it is not refined any longer. Therefore, if the only refinement path to a model m_2 goes through an already discovered complete model m_1 , then m_2 will never be produced.

Later (section 3.4.3 and example 3.6.3 in section 3.6) we shall define a refinement for the class of finite automata. It augments an automaton by adding one transition and, sometimes, a new state. In that particular case we can show that there are many ways to construct a given automaton by applying refinements to an initial, one-state automaton. Then, for any hypothesis, there are many refinement paths by which they can be reached. Therefore each hypothesis will be generated after a finite number of refinement steps, even if some refinement paths to it are stopped by a complete model.

Thus, if every hypothesis is eventually generated, we can prove the following:

Proposition 3.3.1

Let m'_n be the model output by the algorithm after reading n symbols of s .

If there is a bijective function $m \in M$ and a k -random string c such that $s = m(c)$,

Then $\lim_{n \rightarrow \infty} [\text{press}_{m'_n}(s_n) - \text{press}_m(s_n)] = 0$

Proof

To simplify notation, let $c(n) = \text{code}(m, s_n)$ and $c'(n) = \text{code}(m'_n, s_n)$

Note that $s_n = m'_n(c'(n)) = m(c(n))$ Also, since m is bijective, $c(n) = m^{-1}(s_n)$

Since c is k -random, no prefix of it can be compressed more than k bits:

$$(1) K(c(n)) \geq |c(n)| - k$$

On the other hand, one way to compute $c(n)$ is from m^{-1} and s_n . Therefore we can augment (1) into

$$(2) \quad K(s_n) + K(m^{-1}) \geq K(c(n)) \geq |c(n)| - k$$

Since $s_n = m'_n(c'(n))$ is one way to compute s_n , then

$$(3) \quad K(s_n) \leq K(m'_n) + K(c'(n))$$

But $K(m'_n) \leq \text{plex}(m'_n)$ and $K(c'(n)) \leq |c'(n)|$. With these, (3) becomes

$$(4) \quad K(s_n) \leq \text{plex}(m'_n) + |c'(n)|$$

From (4) and (2) we obtain

$$(5) \quad \text{plex}(m'_n) + |c'(n)| + K(m^{-1}) \geq |c(n)| - k$$

Adding $\text{plex}(m)$ to both sides of (5) we get

$$(6) \quad \text{plex}(m'_n) + |c'(n)| + K(m^{-1}) + \text{plex}(m) \geq \text{plex}(m) + |c(n)| - k$$

Let $b = K(m^{-1}) + \text{plex}(m) + k$. This is a constant that does not depend on n .

(6) becomes:

$$(7) \quad \text{plex}(m'_n) + |c'(n)| + b \geq \text{plex}(m) + |c(n)|$$

From here it is easy to see

$$(8) \quad (n - \text{plex}(m'_n) - |c'(n)|) / n \leq (n - \text{plex}(m) - |c(n)|) / n + b/n$$

which is the same thing as

$$(9) \quad \text{press}_{m'_n}(s_n) - \text{press}_m(s_n) \leq b/n$$

Therefore, after a finite time, the perfect model will be among the best hypotheses.



The set H in algorithm 3.3.1 grows exponentially. For a practical implementation, we can add a line that periodically drops those hypotheses that do not achieve a compression factor within a certain threshold of the current maximum. Do we loose completeness by introducing pruning? The above proposition assures that if the perfect model survives the few initial pruning sessions, it will always remain in H , because it will get close enough to the current best. Besides, since it is the perfect model, even if it doesn't achieves maximum compression at every step, it is consistently present in H . The m'_n models are only ad hoc theories, probably different for every n . Still, it is possible for the best model or its refining path ancestors to be discarded, thus, a general pruning algorithm is not complete. Our experiments with finite automata models (see example 3.6.3) show that algorithm 3.3.1 with pruning does produce complete models. They are suboptimal since they differ from the perfect model by some extra states and transitions.

Although a maximum compressibility solution seems so hard to reach, *from a learning point of view we gain something each time we reach a model with a higher compression factor*. Assume that we found a complete model m for the string s with

the following property:

$$(1) (\forall n) \text{press}_m(s_n) \geq q$$

Note that this is a strong assumption, since it requires a minimum compression for every prefix. Some strings may not have such a model. Assume that we were able to parse s_n and we found c_k such that $m(c_k) = s_n$. In virtue of (1),

$$(2) (n - \text{plex}(m) - k)/n \geq q \quad \text{which is equivalent to} \quad 1 - \text{plex}(m)/n - q \geq k/n.$$

From $\lim_{n \rightarrow \infty} (\text{plex}(m)/n) = 0$ we conclude that
($\forall \epsilon > 0$)($\exists N$) such that $n > N \Rightarrow k/n < 1 - q - \epsilon$.

Therefore for every symbol of code the process m will produce at least
 $u = 1/(1 - q - \epsilon)$ symbols of s .

This has an important consequence on the prediction capability of the model m since the above argument proves

Proposition 3.3.2 If $s, c \in \{0,1\}^*$ and $(\forall n) \text{press}_m(s_n) \geq q$,

Then, after a finite prefix, m will always need only 1 bit of information in order to predict the next $\lfloor 1/(1-q) \rfloor$ bits of s .

3.4 Finite Memory Models (FMM)

The idea of this section is to limit a context sensitive grammar to use a finite amount of memory. To this purpose we imagine an automaton whose states are no longer represented by indivisible symbols but by n -tuples or strings of symbols of limited length. This allows for more complex representations. Information common to two states can be represented by common symbols in their strings. Transitions that affect only some symbols of the strings can be represented by schemata that apply to all states that contain those symbols. We shall use this class extensively in section 4.3 since it seems particularly appropriate to model the microworld described there.

Therefore, we introduce FMM, a class of models that can be induced incrementally by continuously refining the hypotheses when they fail to parse the string. The compressibility factor of the newly created hypotheses can be computed without running them on the string prefix seen so far. These models owe their name to the fact that the computations they describe can be made with a finite memory. They are obviously equivalent to finite automata. However we find useful to describe them because they are able to achieve better compression than automata by representing state information in a different way.

Example 3.4.1 Let $L_1(k)$ be the set of Lisp expressions with depth limited to k . $L_1(k)$ is a regular language but its smallest finite automaton representation is larger than an equivalent push-down automata constrained to a limited stack. This is because the latter makes explicit the fact that lists and sublists have the same structure, regularity not captured by the former.

Example 3.4.2 Let $L_2(k) = \{ wa^n w \mid w \in (a+b)^* \text{ and } \text{length}(w) \leq k \}$. This is another regular language that has a more compact representation as a context sensitive grammar constrained to a finite number of applications of certain productions.

Example 3.4.3 Let us attempt to represent by a finite automata the configurations of a finite, discrete space where two objects may be placed and moved independently (say two pawns that can be pushed around a chess board). With the exception of the natural constraint that they cannot be in the same place at one time, the positions of each object can be represented by a separate automaton with 64 states. However, the pair of objects require an automaton with about 2,000 states and their relative independence is no longer explicit in this representation. Nevertheless here is a representation that is shorter and that makes explicit both the independence and interaction of the two objects: *Represent the states as a pairs of symbols, provide transition rules separately for each half of the pair, and add a few rules that affect the entire pair in order to reflect the interaction.*

So, let us stress again the point in defining yet another variation on finite state machines. Finite automata have all the properties (i-vi) required by algorithm 3.3.1 but they can represent only regular languages. Isomorphic subgraphs and other regularities of the transition graph of a finite state machine cannot be made explicit within the normal automata representation. On the other hand, context free and context sensitive grammars are more powerful representations but they are not as easy to induce.

3.4.1 The Class of Models.

Consider a machine with three tapes INPUT, WORK, OUTPUT. The heads on INPUT and OUTPUT are only moving to the right. WORK is a finite length tape. Symbols on the three tapes belong to the finite alphabets I,W,O. The machine can be specified as a set of transition tuples $[a,w,w',b]$ where

- a - the symbol expected on INPUT; it may be null (ϵ)
- w - the expected content of WORK;
- w' - the new content of WORK; it may be null (ϵ)
- b - the symbol to be written on OUTPUT; it may be null (ϵ)

Since WORK cannot contain more than $n = L^{|W|}$ configurations, where $L=|WORK|$, the entire machine is equivalent to a finite automaton with n states. Note that all positions of WORK can be read and written in one transition. A normal Turing machine could simulate this in no more than $2L$ transitions. The machine is deterministic, that is, there are no conflicting transitions (for example $[a,w,w',b]$ and $[a,w,w'',b']$, or $[a,w,w',b]$ and $[\epsilon,w,w'',b']$). To the above constraints we add the fact that the machine cannot go through more than k transitions before writing the next symbol on OUTPUT. We shall keep k constant for simplicity, but the following holds for $k = \text{recursive-function (the length of the output produced so far)}$.

We shall now endow this machine with a compressibility advantage over finite automata. We shall allow transitions be specified by schemata: $[a, \alpha, \beta, b]$ where $\alpha, \beta \in (W \cup V_1 \cup V_2)^*$ are patterns formed with WORK symbols and special variable symbols from two alphabets V_1 and V_2 . Variables from V_1 will match one tape symbol and those from V_2 will match 0 or more contiguous tape symbols. Consequently, a schema will stand for a change operated only on some positions of the tape regardless of the content of the other positions, therefore it will apply to many tape configurations.

3.4.2 Running and Parsing

A snapshot of computation on this machine is a tuple $\sigma = [\alpha w \beta]$ where

α - the INPUT string to the right of the head (the symbols that remain to be read)

β - the OUTPUT string to the left of the head (the symbols written so far)

w - the content of the WORK tape

Take two snapshots $\sigma = [a\alpha \ w \ \beta]$ and $\sigma' = [\alpha \ w' \ \beta b]$. If there is a transition $\tau = [a, w, w', b]$, we say that σ' derives from σ in one step and we write $\sigma \rightarrow_{\tau} \sigma'$ or $\sigma \rightarrow_1 \sigma'$. The special case transitions with null components apply as follows:

$\tau = [\epsilon, w, w', b] \Rightarrow [a\alpha \ w \ \beta] \rightarrow_{\tau} [\alpha \ w' \ \beta b]$
(no input needed)

$\tau = [a, w, \epsilon, b] \Rightarrow [a\alpha \ w \ \beta] \rightarrow_{\tau} [\alpha \ w \ \beta b]$
(no change in the work tape)

$\tau = [a, w, w', \epsilon] \Rightarrow [a\alpha \ w \ \beta] \rightarrow_{\tau} [\alpha \ w' \ \beta]$
(no output produced)

A derivation with at most k steps will be called a k -derivation. The machine starts with an initial snapshot and runs until no new snapshot can be derived from the current one or no output was produced in a k -derivation. With these definitions we can define a function $\text{run}(\sigma, k, b)$ that returns either a snapshot σ' or a special symbol \perp , such that

$\sigma' = \text{run}(\sigma, k, b) \Leftrightarrow$
 $\sigma \rightarrow_{\kappa} \sigma'$ and b was the only symbol written on OUTPUT during this derivation

$\perp = \text{run}(\sigma, k, b) \Leftrightarrow$
the machine stopped within k steps or
a symbol different than b was written on OUTPUT within k steps or
no symbol was written on OUTPUT within k steps

Note that the function $\text{run}(\sigma, k, b)$ checks whether b can be produced within k steps from σ .

Assume now that we have a model M represented as a set of transitions. We have to specify how we can use M to parse a string s . For every character a of s we have to find an input string c such that M reads c and outputs a in at most k steps. For this purpose we have to run M in reverse. Since there might be several ways to parse the same string we have to maintain a set of current snapshots as a way to consider several computations in parallel.

Algorithm 3.4.1

1. Let σ_0 be the initial snapshot with an empty INPUT and a blank WORK tape.
2. $CURRENT = \{\sigma_0\}$
3. Loop until $CURRENT$ is empty
4. Forall $\sigma = [\alpha \ w \ \varepsilon] \in CURRENT$
5. Create $|I|^k$ snapshots $\sigma_j = [\alpha \alpha_j \ w \ \varepsilon]$,
by adding all possible input strings α_j of length k to INPUT of σ
6. Replace σ by these new snapshots
7. Let b be the next symbol to parse
8. Forall $\sigma = [\alpha \ w \ \varepsilon] \in CURRENT$
9. If $\sigma' = \text{run}(\sigma, k, b)$ then replace σ with σ' in $CURRENT$
10. If $\perp = \text{run}(\sigma, k, b)$ then remove σ from $CURRENT$
11. Forall the remaining snapshots in $CURRENT$
12. Remove the INPUT symbols that were not read (if any)

If, during a run through the loop, the expected symbol was not written on OUTPUT, then M failed and the parsing stops. Otherwise, the successful snapshots have on INPUT, to the left of the reading head, the code for the prefix of s parsed so far.

3.4.3 Refinement.

Assume that a model M was able to parse a prefix u of a string s , but failed on the next character, a . That is there is no input string c such that σ_0 with c on INPUT can derive a snapshot with ua on OUTPUT. In this case we would like to expand M into a set of models M' . Since M' has to behave like M on strings that M parsed, we cannot modify or remove transition tuples of M , but we can only add new tuples.

Without loss of generality we can take I to be $\{0,1\}$. Let $\sigma_0 = [\varepsilon, w_0, \varepsilon]$ be the ending snapshot of a derivation that produced u . Let x stand for a 0 or a 1. We know that all k -derivations from σ_0 failed to produce a . Let D be one of these k -derivations. Let $\sigma_i = [\varepsilon, w_i, \varepsilon]$ be the i^{th} snapshot in D . We can distinguish the following cases:

- (a) There is no transition that matches w_i .
 σ_i is the end of D.
- (b) There is a transition $[\epsilon, w_i, w_{i+1}, \epsilon]$ and $\sigma_i \rightarrow \sigma_{i+1} = [\epsilon, w_{i+1}, \epsilon]$.
- (c) There is a transition $[\epsilon, w_i, w_{i+1}, b]$.
 σ_i is the end of D.
- (d) There are both transition $[x, w_i, w_{i+1}, \epsilon]$ for $x=0$ and $x=1$.
By one of them $\sigma_i \rightarrow \sigma_{i+1} = [\epsilon, w_{i+1}, \epsilon]$.
- (e) There are both transition $[x, w_i, w_{i+1}, b]$ for $x=0$ and $x=1$.
 σ_i is the end of D.

In all of the above cases σ_i is a point of failure since it is a step in a k-derivation that failed to produce a. In cases (d) and (e) nothing can be done because no transitions matching w can be added without making M nondeterministic. However, in cases (a), (b) and (c) new transitions can be added, therefore enabling D to continue in new ways after the i^{th} snapshot. The new transitions are:

- (a1) Add $[\epsilon, w_i, w, \epsilon]$
- (a2) Add $[\epsilon, w_i, w, a]$
- (a3) Add $[0, w_i, w, \epsilon]$ and $[1, w_i, w', \epsilon]$
- (a4) Add $[0, w_i, w, a]$ and $[1, w_i, w', \epsilon]$
- (b1) Replace $[\epsilon, w_i, w_{i+1}, \epsilon]$ by $[0, w_i, w_{i+1}, \epsilon]$ and $[1, w_i, w, \epsilon]$
- (b2) Replace $[\epsilon, w_i, w_{i+1}, \epsilon]$ by $[0, w_i, w_{i+1}, \epsilon]$ and $[1, w_i, w, a]$
- (c1) Replace $[\epsilon, w_i, w_{i+1}, b]$ by $[0, w_i, w_{i+1}, b]$ and $[1, w_i, w, \epsilon]$
- (c2) Replace $[\epsilon, w_i, w_{i+1}, b]$ by $[0, w_i, w_{i+1}, b]$ and $[1, w_i, w, a]$

Each of the 8 modifications above is in fact a multiplicity since w and w' stand for any WORK configuration with the only constraint that $w \neq w'$. We only have to specify how to generate these configurations. If we want to consider all possible machines in the class, we have to be able to enumerate machines with different lengths of the work tape.

Given that any FMM can be seen as a simple automaton, we could approximate its complexity by the number of transitions. However, since several transitions might

be encoded by a single schema, a better approximation is the sum of lengths of all schemata. This is supported by the fact that a FMM can be encoded as a string of schemata (see subsection 3.4.5).

Proposition 3.4.1 The refinement operator just defined has the following properties:

- (i) $\text{refinement}(m, u, a)$ is finite
- (ii) $m' \in \text{refinement}(m, u, a) \Rightarrow \text{plex}(m) \leq \text{plex}(m')$
- (iii) there is a null model m_0 ($|Work|=0$, and no transitions) and $\{m_0\}$ is a lower bounding set.
- (iv) the refinement operator is complete
- (v) $\forall m' \in \text{refinement}(m, u, a)$, the state of m' after parsing ua is known

Proof

(i) There are a finite number of failing snapshots to which we add transitions. To each snapshot we add at most 2 types of transitions (see a1,2,3,4 b1,2 c1,2). For each transition type we have a transition for each resulting configuration of the work tape. Since at each refinement we increase the length of $WORK$ with at most one, there are a finite number of resulting configurations.

(ii) Each model in $\text{refinement}(m, u, a)$ has one more transition than m . This extra transition may remain as such, or it may be absorbed in an existing transition schema, or it may pair with another to form a schema. In every case the number of transitions or schemata cannot decrease.

(iii)(iv) One can easily construct any given FMM by adding transitions to the null model.

(v) Let us look again at how refinement is done. Let $\sigma_0 = [\epsilon, w_0, \epsilon]$ be the ending snapshot of a derivation that produced u . We know that all k -derivations from σ_0 failed to produce a . Let D be one of these k -derivations. Let $\sigma_i = [\epsilon, w_i, \epsilon]$ be the i^{th} snapshot in D .

In refinement cases a2,a4,b2,c2 we add a transition that outputs a and changes the tape configuration to w . The refined machines can use this new transition to parse a and remain with tape configuration w .

In refinement cases a1,a3,a4,b1,c1 we add a transition that has no output and changes the tape configuration to w . Since this new transition applies to the i^{th} snapshot in the derivation D and goes to a new snapshot $\sigma_{i+1} = [\epsilon, w, \epsilon]$, we should investigate all derivations from σ_{i+1} until either a is produced or $k-i-1$ steps are consumed.

Those resulting in the production of a are successful parsings and the resulting snapshots are known.

3.4.4 Incremental Computation of Compressibility.

In order to evaluate the compressibility achieved by FMM hypotheses while parsing a string we have to add to each work tape configuration (i.e. to each state) a counter that stores how many times the model reached that configuration. When a new configuration is created by refinement, its counter is initialized to 1. To each model we also attach a variable called `code_length` that contains the length of input read by it during the parsing process. The counters allow us to evaluate the `code_length` of the models newly created by refinement, without running them:

Assume that m' was created from m by refinement, that is by adding a transition to a configuration q of m . Suppose that m had one transition that apply to q and m' has two transitions that apply to the corresponding q' . Note that all states of m' inherit their counter values from m since m' would have used the same state sequence for parsing. Also suppose that length of the segment parsed sofar is k . Had we used m' to parse the string sofar, each time we executed a transition out of q , a one bit input would have been necessary to distinguish between the two transitions. Therefore,

$$(4) \text{ code_length}(m') = \text{code_length}(m) + \text{counter}(q)$$

The above procedure allows us to know $\text{press}(m)$ at each parsing step:

$$(5) \text{ press}(m) = (k - \text{code_length}(m) - \text{plex}(m)) / k$$

3.4.5 Computation of Transition Schemata.

The refinement method outlined above creates specific transitions. Let us now concentrate on how transition schemata emerge from this.

(i) Each time we add a new transition $[i, w, w', o]$ we need a new configuration w' . We choose w' out of all possible 2^k strings of length k . When these are exhausted, we increment k . Assume that we have a function `min_gen` that finds the minimal common generalization of two schemata.

$$(6) [w, w'] = \text{min_gen}([w_1, w_1'], [w_2, w_2'])$$

We compute the common minimal generalization between the new transition and all other schemata and we replace the new transition with the minimal generalization

that causes the smallest increment in the complexity of the model. Note that this is obviously a greedy approach to finding the best set of schemata equivalent to the primary transitions. Therefore it might not achieve the minimal encoding of the model by schemata.

(ii) (Another way) We generate new transitions in increasing order of complexity. Each time we add a new transition $[i, w w', o]$, we choose w' to differ from w by only one position, then by two, etc.

3.4.6 Composition

It is possible to find a complete model for a string that doesn't capture all regularities of that string and, consequently, does not achieve the highest possible compression. An idea that we shall develop later (in section 3.6) is to use the code produced by that model as input for a new induction problem. In other words, once we find c and $m \in M$ such that $s = m_1(c_1)$ with $\text{press}_s(m_1) > 0$, we can start looking for a model for c_1 . If we find c_2 and $m_2 \in M$ such that $c_1 = m_2(c_2)$ with $\text{press}_{c_1}(m_2) > 0$, then we are in possession of a better model for s , namely $m_1(m_2(c_2))$.

Direct induction on s might have eventually produced the composition $m = m_2 \otimes m_1$ provided the class M is closed under composition. This would have required more effort since the algorithms that we consider require time and memory exponential in $\text{plex}(m)$. We can show in the remainder of this section that FMM's are closed under composition, therefore induction of complex models may proceed stepwise.

We could view FMMs as machines with one tape that has the following structure:

$$(7) \quad o_0 o_1 \dots o_k \quad w_k \quad i_{k+1} i_{k+2} \dots$$

where $o_0 o_1 \dots o_k$ are symbols output so far, w_k is a string representing the content of the work tape, and $i_{k+1} i_{k+2} \dots$ are the input symbols that remain to be read. Transitions would take the form

$$[w_k i_{k+1} \rightarrow o_{k+1} w_{k+1}].$$

A computation would slide the working segment w from the left end of the tape to the right end of it:

$$(8) \quad w_0 \ i_1 \ i_2 \ \dots \ i_n \ \rightarrow \dots \rightarrow \ o_1 \ \dots o_k \ w_k \ i_{k+1} \ i_{k+2} \ \dots i_n \rightarrow \\ \rightarrow \ o_1 \ \dots o_k \ o_{k+1} \ w_{k+1} \ i_{k+2} \ \dots i_n \rightarrow \dots \rightarrow \ o_0 \ o_1 \ \dots o_n \ w_n$$

Note that the indices in (8) suggest, only for the purpose of illustration, that one input is read and one output is produced at every step. This need not be the case. However, from (8) one can see how we could feed the output of this machine into another FMM. Both of them can work on the same tape, the second following the first. If we ask the first machine to wait for the second to consume the most recent output symbol that the first has produced, the tape will evolve this way:

$$(9) \quad o_0 \ \dots o_k \ w_k'' \ w_k' \ i_{k+1} \ i_{k+2} \ \dots \rightarrow \ o_0 \ \dots o_k \ w_k'' \ x \ w_{k+1}' \ i_{k+2} \ \dots \rightarrow \\ o_0 \ \dots o_k o_{k+1} \ w_{k+1}'' \ w_{k+1}' \ i_{k+2} \ \dots$$

Note that w'' closely follows w' and their concatenation, including the occasional symbol x , can be seen as a larger work segment w . Therefore (9) also illustrates the computation steps of an equivalent FMM with $|w| = |w'| + |w''| + 1$. But the composition of FMM's can also be used in reverse, that is, in parsing. This allows a stepwise induction for strings that accept compositions of FMM's as models. More about it in section 3.6.

3.5 Induction of Submodels: Libraries

3.5.1 The Class of Models.

So far we have searched only for complete models. Let us now consider the case where we organize several regularities of a string s into a library L of routines such that each routine can generate, with appropriate input data, some fragments of s . We shall look now at a new class of models that consist of a library of routines and a main program that is only a sequence of routine invocations.

Note that, when the regularities in L do not cover the entire string, we can always add to L the identity function in order to turn the partial model into a complete one. Also, parsing with such a model requires unique decyphering, that is, at any point we have to be able to determine which routine applies.

When computing the gain and compressibility of such a model we have to take into account the size of reference, that is the way to distinguish among the routines in L . As a first approximation we can take this to be $\log|L|$. A better approximation is the entropy of the frequency of usage distribution.

$$\text{gain}(L) = \sum_{r_i \in D} (\sum_j (|w_{ij}| - |\text{reference}_i| - |\text{code}_{ij}|) - \text{plex}(r_i))$$

where $r_i(\text{code}_{ij}) = w_{ij}$

Let us illustrate the above formula through several examples where we have to parse a string s from $(a+b)^*$. We start with two extreme examples.

Example 3.5.1 Routines without arguments. Library = $\{ r_0, r_1, r_2, r_3, \}$ where $r_0()=aa, r_1()=ab, r_2()=ba, r_3()=bb$. Since there are no arguments, $|\text{code}_{ij}|=0$ for all routines. Since there are 4 routines $|\text{reference}_i| = \log 4 = 2$. Also $|w_{ij}| = 2$ for all parsed segments. *Therefore there is no positive gain.*

The only regularity we could take advantage of would be a *nonuniform probability distribution* on the routines and, consequently, for the 4 words aa, ab, ba and bb . In that case $|\text{reference}_i| = -\log(p_i)$ and the average gain per parsed segment is $\text{gain}(L) = 2 - H(p)$ where $H(p)$ is the Shannon entropy of the distribution. That is $\text{gain}(L) = \text{MaxH} - H(p)$. In the data compression literature [Lempel-77,78][Rissanen-83] such string routines are called *block codes* since the compression process translates a contiguous block of symbols from the source stream into a code - here: the routine reference.

Example 3.5.2 A one routine library $L = \{ r(c) \}$ where $r(0)=aa$, $r(1)=ab$, $r(2)=ba$, $r(3)=bb$. Now $|code_{ij}|=2$ and $|reference_i|=0$, hence no positive gain. All the other considerations from the previous example apply here too.

Example 3.5.3 [Caianiello-89] defines two types of library models for finite strings: *codes* and *classifications*. Codes are block codes as in example 3.5.1 and classifications are partitions of the alphabet into classes of symbols. A library corresponding to a classification would have a routine for each class and routine arguments would distinguish among symbols within one class. Codes and classifications can be composed in both directions to generate more complex routines: classes of strings and strings of classes.

Example 3.5.4 [Lempel-76,77,78] describe a now famous text compression algorithm (LZ), that creates block codes incrementally. Since here routines stand for strings, we shall treat them as strings. Let s be the string to be compressed (parsed). The algorithm admits routines to be prefixes of other routines. In order to maintain the unique decyphability condition, the routine chosen at each parsing step is the one that matches the longest prefix of s . This is an outline of the algorithm:

Initialize the library to one routine identical to the empty string.

Loop

Let r be the longest routine in the library that matches the prefix of s .

Then s can be written rcs'

where r is the prefix, c the next character and s' the remainder of the string.

Create a new routine $r' = rc$

Code r' by the pair $[reference(r), c]$

Replace s by s'

For a string generated by a Markov source the compression algorithm should capture and take advantage of the dependencies of each symbol from the preceding context. In other words, if symbol 'a' follows the string w , that is appears in the context w , then it should be assigned a code length proportional with $-\log P(a|w)$. For a finite string such probabilities can be collected in a first pass over the string, and used to compress it in a second pass. The LZ algorithm is incremental, therefore it only approximates those statistics. It assigns to 'aw' a code length of $1+|reference(w)| = 1+\log(|D|)$. The algorithm asymptotically achieves the compressibility of an optimal variable-to-block encoder. For infinite strings generated by a stationary ergodic source, the per symbol compression converges to the per symbol entropy of the source.

Example 3.5.5 Let $S = \{ x_1, \dots, x_n \}$ be a set of points / vectors in an n -dimen-

sional space. Given a fix point x_0 , S can be expressed as $S' = \{y_1, \dots, y_n\}$ where $y_i = x_i - x_0$. This is equivalent to defining a routine r_{x_0} such that $x_i = r_{x_0}(y_i)$. Assume for simplicity that the coordinates in this space are natural numbers. We can approximate $|x_i|$, the representation length for a vector x_i in terms of the representation length for its numeric coordinates. The gain of representing S by r_{x_0} is

$$\text{gain}(S, r_{x_0}) = |S| - |S'| - |\text{reference}(r_{x_0})| = \sum_i |x_i| - \sum_i |y_i| - |x_0|$$

A partition of S into the sets $C_j, j=1, \dots, k$ together with a set of points $\{x_{01}, \dots, x_{0k}\}$ with the property

$$(\forall j=1, \dots, k) \text{ gain}(C_j, r_{x_{0j}}) > 0$$

is in fact a clustering of S . This is not surprising, since finding any good clustering for S is equivalent to discovering a part of its structure. Let $H(x_i|C_j)$ be the information needed to retrieve $x_i \in C_j$, knowing C_j . A good clustering would minimize

$$\sum_{j=1, k} (|C_j| + \sum_{(x_i \in C_j)} H(x_i|C_j))$$

which is equivalent to maximizing the gain of representing S by the library of routines $\{C_1, \dots, C_k\}$.

Example 3.5.6 Let us look again at a set of points in an n -dimensional space. In the previous example we considered clusters defined by a central point. This entails very simple routines. The flexibility of libraries allows more complex examples. Imagine $S = \{x_1, \dots, x_n\}$ as a cloud of points arranged around a certain curve C . Each can be expressed as $x_i = y_i + z_i$ where z_i is a proximal point on C . But, z_i can be reduced to the equations for C and a scalar parameter ζ_i . Therefore a more complex routine can reduce all x_i to $r(z_i, \zeta_i)$.



Libraries are a class of models. They does not restrict in any way the type of the component routines, they can be essentially anything. Besides, behavior of routines can be context dependent, they may convey information to each other, but all this is controlled by the sequence of invocations and routine parameters. A sequence of invocations can be represented as a string of pairs $\langle \text{routine_index}, \text{routine_arguments} \rangle$. Any interdependency of routines will show up as a regularity of that sequence and it does not have to be captured by the library itself.

Still, we have to show how can we parse with libraries and how can we induce libraries. Intuitively, if we use the analogy to clustering, then Chaitin's mutual information, $K(X:Y)$, can be seen as a sort of distance. For instance, we can use $K(X:Y)$ to define a normalized, symmetrical measure for algorithmic similarity:

Definition 3.5.1 algorithmic similarity:

$$\sigma(X,Y) = K(X:Y) / K(X,Y) = (K(X) + K(Y)) / K(X,Y) - 1$$

One can verify that

- (i) $\sigma(X,Y)$ is symmetrical
- (ii) $1 > \sigma(X,Y) \geq 0$
- (iii) $\sigma(X,X)$ is ,generally, very close to 1.

3.5.2 Fragmentation, Parsing and Refinement

Assuming that we have a parsing procedure for each routine, we have to decide which routine in the library, if any, should parse the current string fragment. In example 3.5.5 the analyzed structure was a set of substructures. In such a case the next fragment to be parsed was *well defined* - one element of the set, i.e. a point. When the next fragment is well defined all routines can be tried on it, one after another. Concept formation through clustering attribute-value vectors [Gennari-89] also has well defined fragments. At any given time the algorithms will have to deal only with the next attribute values vector. Texts and visual images, however, do not have well defined fragments. A good example of difficult fragmentation is listening to announcements spoken over a loudspeaker in a foreign airport. Even without knowledge of the language we should be able to notice if our name or our destination show up in an announcement. Another is offered by reading of handwriting. Each routine in a library should encode the knowledge of a certain substructure. Fragmentation becomes, for each routine, the problem of recognizing an instance of that structure and decide its boundaries. Therefore *fragmentation will be determined by parsing*.

Here is a simple approach. We start parsing with all routines in parallel. Eventually they will stop, each one, probably, after parsing a different prefix. (If one doesn't stop, then that one alone is a complete model for the string and the library is a degenerate one.) The routine that obtains the highest compression factor wins. If fragments are not known a priori, then the winning routine also defines the current fragment as being the longest prefix parsed by it. A Bayesian criterion for fragmentation would chose a routine m by maximizing $P(mlw)$. In section 2.4.1 we proved that this is equivalent with minimizing $K(m) + K(w|m)$. But our approximations for $K(m)$ and $K(w|m)$ are, respectively, $\text{plex}(m)$ and $\text{code}(m,w)$. Since $\text{press}_w(m) = (|w| - \text{code}(m,w) - \text{plex}(m)) / |w|$, the criterion becomes $\max \text{press}_w(m)$, supporting our approach.

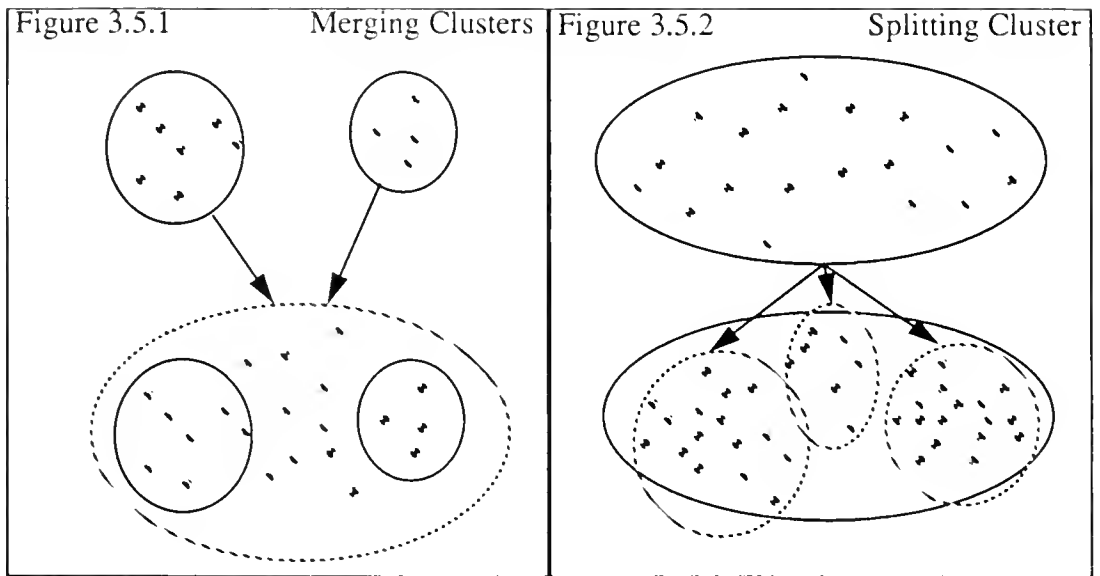
If a class M of models is recursively enumerable, then libraries, that is, finite sets of such models are also enumerable. Hence induction by enumeration will work as well on libraries. A refinement operator available on the class of models also induces a refinement operator on libraries. A library can be refined by refining one of the component models on its current point of failure. If we measure the complexity of a library as the sum of the complexities of its components, the library refinement will inherit all the desired properties (i-v in section 3.3) from the component model refinement.

A population of library models could participate in a *genetic search*. In this case the routines provide natural cutting points, and *crossing* amounts to exchanging routines between libraries.

A library operates a *classification on fragments*. Every routine in a library defines a *class* (or cluster) that contains all fragments parsed by it. The routine captures a part of the common algorithmic information shared by those fragments, ideally, their structure. *Therefore libraries are theoretical models for classification*. A partial model for a finite amount of data may turn out not to be a basic regularity in view of further data. Research in conceptual clustering, (for example, the COBWEB and CLASSIT algorithms surveyed in [Gennari-89]) encountered the problem and described an evolution of clusters through (a) *merging* or (b) *splitting*.

a) Two clusters that were initially well separated may merge if new points are added in the space between them. (figure 3.5.1) Routines should adapt as well. For some representations (vectors, first order formulae, etc.) merging can be defined as a minimal common generalization. For other representations this is an open problem. For finite data, if all the past fragments are available, they could be used for induction of a common model.

b) A cluster may group some initial uniformly distributed points, but when new points are added, new concentrations may appear, and the cluster may be split into smaller groups. (figure 3.5.2) A routine r maps all fragments $\{x_i \mid i \in I\}$ parsed by it into their codes c_i . If the codes c_i can themselves be compressed by a library $\{p, q\}$, that is, $c_i = p(d_i)$ for $i \in I_1$ and $c_i = q(e_i)$ for $i \in I_2$, then r can split into the pair $r_1 = p \otimes r$ and $r_2 = q \otimes r$ (where \otimes stands for composition of functions). This operation can be interpreted as specialization since the classes defined by r_1 and r_2 are contained in the class defined by r . In many approaches of learning from examples, specialization of hypotheses is prompted by negative examples. Here the more specific r_1 and r_2 are generated as a result of positive examples ! We shall use this distinction in section 5.2 to present explanation-based generalization as a specialization.



Some obvious issues are left out. Splitting, merging, and growing to incorporate new points, can be easily defined for clusters of points. However, for routines, this an open problem. We leave these questions unanswered here, but we shall investigate them in a particular case (end of section 4.3.3) where the routines are finite automata.

3.6 Stepwise Induction

The main question here is to what extent could a learning program discover the structure of a sequence in small steps. An experiment reported by [Gerwin-74] is relevant here. A group of students had to resolve a case of induction of real functions of one variable. They had to guess analytical formulae for unknown functions, given noisy sets of argument-value pairs $[x, f(x)]$. The subjects were told that target functions were arithmetic combinations of simple analytical functions (sin, cos, square, exp, log etc.). The following problem solving protocol was observed. The subjects advanced a hypothesis for the main component $h(x)$ of the function (say $\sin(x)$ in a periodical function), and then computed a residual data (ex. $[x, f(x)-h(x)]$ or $[x, f(x)/h(x)]$) that is, a part that was not accounted for by that component. The method would then be recursively applied on the residual data if that displayed less variance than the original one.

Gerwin's paper raises the more general question: In what circumstance can an already found partial regularity be *subtracted* from the data in order to simplify the induction problem? What kind of structures can be discovered step by step? At first let us take a look at those structures that are constructed by assembling many instances of a few types of building blocks. Lego is a typical example, as well as city maps that are made up of house blocks, street segments, plazas, intersections and parks. These are instances of the *natural spaces* introduced informally in chapter 1. We would like to generalize the idea of building block as a special type of regularity.

We recall (Definition 2.2.8) the d-diameter complexity of an object X:

$$K_d(X) = \min [K(\alpha) + \sum_i K(X_i)]$$

where $\{ X_i \mid \text{diameter}(X_i) \leq d \}$ is a partition of X

that is, the minimum length needed to describe X as an assemblage of parts, each not greater than d. All partitions of X into such objects, X_i , are considered, together with α , the description of how should they be assembled. All pieces should have a separate descriptions without cross-references. [Chaitin-78] suggests that a study of $u(d)=K_d(X)-K(X)$ when d varies between 0 and the diameter of X. First let us note that

$$(a) \quad d \geq |X| \Rightarrow u(d) = 0$$

since for parts larger than $|X|$, $K(X)$ itself is the minimum description length

$$(b) \quad d_1 \geq d_2 \Rightarrow u(d_2) \geq u(d_1)$$

since as d grows, we can take advantage of more complex substructures and more distant correlations

Changes occur at the points where d exceeds the diameters of important sub-structures of X . Therefore $K_d(X)$ provides a kind of *spectrogram* of the object X .

An ideal vocabulary of building blocks would contain elements as different from each other as possible, since any substantial amount of shared information among blocks could support a smaller vocabulary. Therefore they should be algorithmic independent. This allows us to define a criterion for their identification:

Proposition 3.6.1 If a structure is composed from highly algorithmic-dissimilar constituents, then those constituents can be identified, during model refinement, by local maxima of compressibility as a function of model complexity.

Argument Let w_1, w_2, w_3 be fragments of a string s . Let m be a model for w_1 but not for w_2 or w_3 . We assume that w_1 and w_2 are algorithmically independent (dissimilar). Their mutual information is very small. Thus we can expand a minimal program for w_1 , in order to produce w_2 , but the addition will be almost as large as a minimal program for w_2 alone. Also, let w_1 and w_3 have a similar structure. Thus program for w_1 needs only a small addition to generate w_3 , since the mutual information of w_1 and w_3 is almost the same as the total information of w_3 . The two assumptions can be expressed as:

$$(a) \ K(w_1 : w_2) < \epsilon \iff K(w_2 | w_1) > K(w_2) - \epsilon$$

$$(b) \ K(w_3 | w_1) < \epsilon \iff K(w_3 : w_1) > K(w_3) - \epsilon$$

Let m' be a minimal refinement of m that parses $w_1 w_2$. Unless $\text{plex}(m') = \text{plex}(m) + K(w_1 | w_2)$, that is unless m' is substantially more complex than m , m' will not be able to maximally compress w_2 . But (a) makes such a refinement unlikely. Let us look at one of the smallest additions that could take m out of impasse.

This is the *copy* program, that prints its input to the output, coding every string by itself. Let m' be m endowed with *copy*, and let us estimate its performance on $w_1 w_2$. If w_2 is coded by itself, then $|\text{code}(m', w_1 w_2)| = |\text{code}(m, w_1)| + |w_2|$.

$$\begin{aligned} \text{press}_{w_1 w_2}(m') &= (|w_1 w_2| - |\text{code}(m', w_1 w_2)|) / |w_1 w_2| = \\ &= (|w_1 w_2| - |\text{code}(m, w_1)| - |w_2|) / |w_1 w_2| = (|w_1| - |\text{code}(m, w_1)|) / |w_1 w_2| = \\ &= (|w_1| / |w_1 w_2|) \text{press}_{w_1}(m) = \text{press}_{w_1}(m) / |w_2| < \text{press}_{w_1}(m) \end{aligned}$$

Therefore m' can parse anything but can compress less. Its performance

decreases with the length of w_2 and assumption (a) forces w_2 to be a relatively long string. (If it were a short one, its complexity would be small, hence it would have been derivable by a program for w_1 with a small addition).

To parse w_3 , m has to be refined too. But (b) shows a small addition to m yields a high compression performance on w_1w_3 . Certain classes of models may not allow expression of that small addition as a refinement. For example, a recursive invocation is a small addition not expressible in a finite automata class.

Nevertheless this discussion shows that *there is a trade-off point beyond which models should not be refined any more*. If the encountered novelty w_2 is small (that is, similar to the model), then it is likely to be incorporated into the model without a serious loss in compressibility. If the novelty is large, all small additions to the model will degrade compressibility and looking for a different model is a better solution.



Note that assumption (a) forces w_2 to be a relatively long string. If it were a short one, its complexity would be small, hence it would have been derivable by a program for w_1 with a small addition. The compression factor of m' would only suffer a small decrement. If w_2 is long, it means that s has in fact two components that are algorithmically substantially different. One is w_2 , the other is the pair w_1, w_3 . Therefore plotting the compression factor of m' during parsing the string $w_1w_2w_3$, we would see it decreasing along w_2 and increasing again w_3 .

This principle justifies the following definition of *basic regularity*. Let us consider a refinement that increases complexity by a small quantum. Let $m \ll m'$ denote that m' is obtained from m through one or more refinement steps. Let $0 < \lambda < 1$ be a real number.

Definition 3.6.2 a λ -*basic regularity* of s is a model m with the property

If $m' \gg m$ & $\text{press}_s(m') \geq \text{press}_s(m)$

Then $\exists m''$ such that $m' \gg m'' \gg m$ & $\text{press}_s(m'') \leq \text{press}_s(m) - \lambda$

The parameter λ affects the separation between regularities. A higher λ forces a higher separation. Using a higher λ , we should be able to identify highly independent building blocks, if there are any.

Example 3.6.1 [Caianiello-71] describes an algorithm (Procrustes) to analyze finite Italian texts. The algorithm develops a block code library model by refining one letter hypotheses. It starts with a library identical to the alphabet of the text. For every element x_i of the library, it considers the refinement $x_i \rightarrow x_i y_j$ for every letter y_j that follows x_i in the text. The decision to stop refining the routines is based on the relative size of the following information measures:

$$(1) H(Y) = - \sum_j P(y_j) \log P(y_j)$$

the average information needed to specify a symbol y

$$(2) H(Y/x_i) = - \sum_j P(y_j|x_i) \log P(y_j|x_i)$$

the average information needed to specify a symbol y following an occurrence of x_i

$$(3) H(Y/X) = \sum_i P(x_i) H(Y/x_i)$$

the average information necessary to specify a symbol y following an occurrence of any routine

Two criteria to decide when to refine a routine x_i are investigated:

$$(a) H(Y/x_i) < T$$

where the threshold T is chosen such that $0 \leq T \leq H(Y)$, that is, x_i substantially determines the symbol following it in the text

$$(b) H(Y) - H(Y/x_i) > H(Y) - H(Y/X)$$

that is, the information gain in guessing what follows x_i is larger than the average one

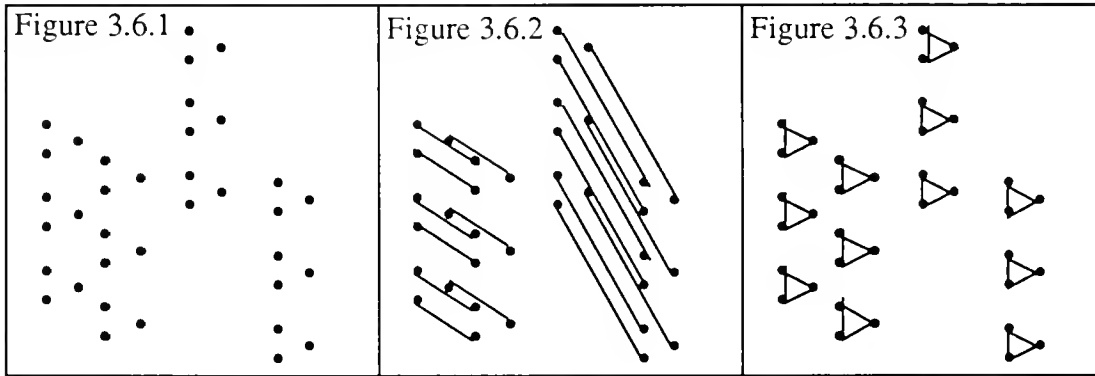
In each case the algorithm converges to a library that contains the basic sub-strings of the text. For example, a natural language text yields a set of syllables; a text of Morse code yields the codes of the individual letters.

Example 3.6.2 Readers familiar with low level, preattentive visual processing will recognize this as an instance of perceptual grouping. It seems that many criteria for grouping, among which, those highlighted by the Gestalt theory (proximity of points, colinearity, bimodal textures, shape vs. ground), are explained by some form of minimal length encoding.

The structure to analyze is a set of 36 points in a two dimensional space (figure

3.6.1). They are given as a set of pairs (x_i, y_i) each coordinate being an eight bit integer. We are considering compressing this description with the following class of models R . A routine $r \in R$ is defined as a set of points (vectors): $r = \{ p_1, \dots, p_k \}$ and has one argument p of type point. Given an argument p_0 , $r(p_0) = \{ p_0 + p_1, \dots, p_0 + p_k \}$ where $(x_0, y_0) + (x_i, y_i) = (x_0 + x_i, y_0 + y_i)$. Also $|r| = \sum_i |p_i|$. Note that all the occurrences of one routine are one configuration of points translated to different positions. For this structure we shall study $\text{gain}(d)$ by considering, in turn, routines with increasing numbers of points.

At each complexity d we shall try to minimize the number of routines in the library L_d , since each one achieves the same compression: coding d points into one for each occurrence. If for some values of d we cannot parse all points with routines of size d , we shall accept in the library a complexity one routine $r_1 = \{(0,0)\}$. This allows a point p not parsed by other routines to be represented as $r_1(p)$. Since the combinatorial complexity was not overwhelming, the following decompositions were worked by hand.



For $d=2$, $L_2 = \{r_2', r_2''\}$ (figure 3.6.2) each routine having 9 occurrences. Since there are 2 routines, the size of reference is 1 bit.

$$\text{Gain} = 36 \times 16 - 9 \times (16+1) - 2 \times 16 - 9 \times (16+1) - 2 \times 16 = 206$$

For $d=3$, $L_3 = \{r_3\}$ (figure 3.6.3) with 12 occurrences. The size of reference is 0.

$$\text{Gain} = 36 \times 16 - 12 \times 16 - 3 \times 16 = 21 \times 16 = 336$$

For $d=4$, $L_4 = \{r_4\}$ with 9 occurrences (figure 3.6.4: for clarity, only 3 out of 9 occurrences are shown). The length of reference is 0.

$$\text{Gain}(4) = 36 \times 16 - 9 \times 16 - 4 \times 16 = 23 \times 16 = 368$$

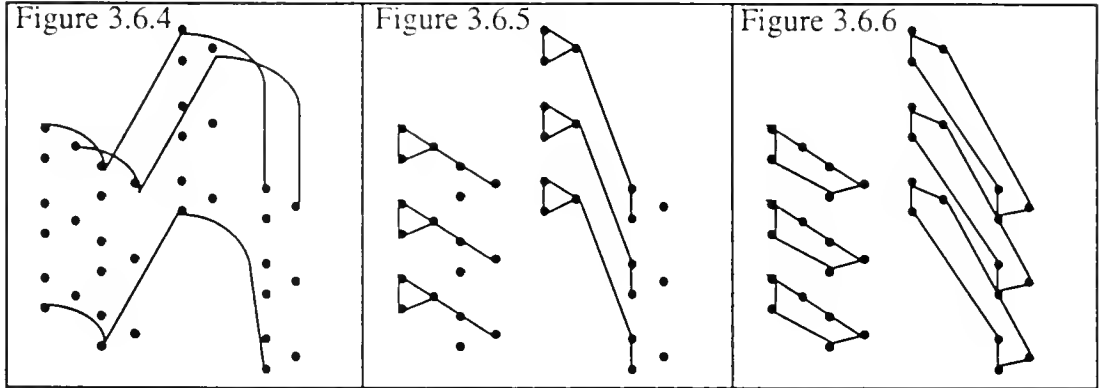
For $d=5$, $L_5 = \{r_1, r_5', r_5''\}$ with 6,3,3 occurrences respectively (figure 3.6.5).

The length of reference is 2.

$$\text{Gain}(5) = 36 \times 16 - 6 \times (16+2) - 1 \times 16 - 3 \times (16+2) - 5 \times 16 - 3 \times (16+2) - 5 \times 16 = 184$$

For $d=6$, $L_6 = \{r_6', r_6''\}$ each with 3 occurrences (figure 3.6.6) The length of reference is 1.

$$\text{Gain}(6) = 36 \times 16 - 3 \times \{16+1\} - 6 \times 16 - 3 \times \{16+1\} - 6 \times 16 = 282$$



For $d=7$, $L_7 = \{r_1, r_7\}$ with 15 and 3 occurrences respectively. The length of reference is 1.

$$\text{Gain}(7) = 36 \times 16 - 15 \times (16+1) - 1 \times 16 - 3 \times (16+1) - 7 \times 16 = 142$$

For $d=8$, $L_8 = \{r_1, r_8\}$ with 12 and 3 occurrences respectively. The length of reference is 1.

$$\text{Gain}(8) = 36 \times 16 - 12 \times (16+1) - 1 \times 16 - 3 \times (16+1) - 8 \times 16 = 177$$

For $d=9$, $L_9 = \{r_9\}$ with 4 occurrences. (figure 3.6.7) The length of reference is 0.

$$\text{Gain}(9) = 36 \times 16 - 4 \times 16 - 9 \times 16 = 368$$

For $d=10$, $L_{10} = \{r_1, r_{10}\}$ with 6 and 3 occurrences respectively. The length of reference is 1.

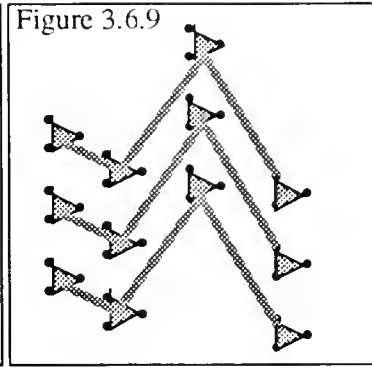
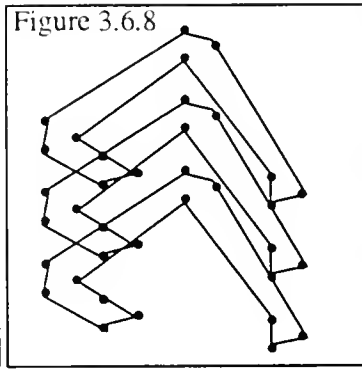
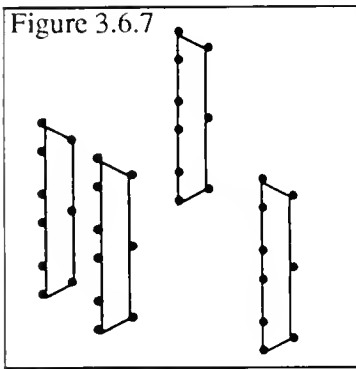
$$\text{Gain}(10) = 36 \times 16 - 6 \times (16+1) - 1 \times 16 - 3 \times (16+1) - 10 \times 16 = 247$$

For $d=11$, $L_{11} = \{r_1, r_{11}\}$ each with 3 occurrences. The length of reference is 1.

$$\text{Gain}(11) = 36 \times 16 - 3 \times (16+1) - 1 \times 16 - 3 \times (16+1) - 11 \times 16 = 282$$

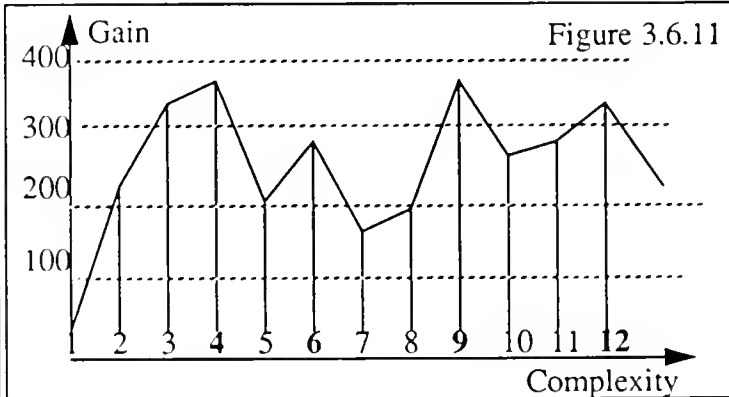
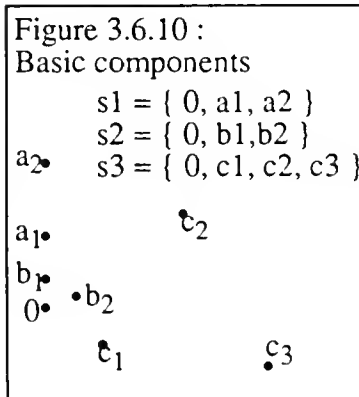
For $d=12$, $L_{12} = \{r_{12}\}$ with 3 occurrences (figure 3.6.8). The length of reference is 0. $\text{Gain}(12) = 36 \times 16 - 3 \times 16 - 12 \times 16 = 336$

We do not continue since it is already apparent that the first local maximum of $\text{Gain}(d)$ corresponds to a basic component of this structure, and the other maxima correspond to important components too (figure 3.6.11) :



plex =	1	2	3	4	5	6	7	8	9	10	11	12
gain =	0	206	336	368	184	282	142	177	368	247	282	336
max				*		*			*			*

Note that this decomposition discovers two of the three basic components of this structure. The entire set of points can in fact be generated as a product of 3 sets of vectors (figure 3.6.10).



If we define $s_1 \times s_2 = \{ v_1 + v_2 \mid v_1 \in s_1 \text{ \& } v_2 \in s_2 \}$ then $s_1 \times s_2 \times s_3$ represents the entire set. But s_2 is equivalent to r_3 and s_3 is equivalent to r_4 . The building blocks s_1, s_2, s_3 are basic in the sense that they are not reducible to a more compact representation through the class of models considered here.

Let us end this example with the following remarks:

- (1) The structure of this set of points is hierarchical. This allows a simpler, stepwise approach. Assume now that we stop searching for models after we found the first local maximum, that is, at r_4 . This compresses our initial structure to a code. This code is in fact a group of 9 points, one for each occurrence of r_4 .

We restart now the process of enumerating models to see whether this code can be compressed any further. We find r_3 as a basic regularity. We have now a better model that consists of $\{r_4\}$ composed with $\{r_3\}$. (figure 3.6.9) In this case we say that our structure has two levels of regularity.

(2) $\{r_3\}$ happens to be a basic regularity but, although it is the best model in its complexity class (3 points), it is not a local maximum in figure 3.6.10. Note however that we did not plot compressibility along a refinement path - as required in definition 3.6.2 - but we simply took the best model for each complexity value.

Assume that we define a minimum refinement of a routine as the addition of one point to the set. $\{r_3\}$ could be refined into many 4 point sets. It is easy to see that $\{r_3\}$ conforms to definition 3.6.2, since all these minimum refinements of it achieve less compressibility.



The concept of basic regularity suggests two approaches to the induction of very complex structures:

- (a) *the construction of a library model where each routine is a basic regularity;*
- (b) *the construction of a complete model that is a basic regularity, followed by the attempt to compress the code*

Definition 3.6.3 A structure that is compressed by a library type model or a composition of models is a hierarchy.

Proposition 3.6.2 Although most of the interesting learning problems are intractable, hierarchical structures may be inducible in a practical context. On a k level structure where each level has complexity less than d , a stepwise induction algorithm will take $O(k \cdot \exp(d))$ steps.

Example 3.6.3 A regular expression with a hierarchical structure. We report here an experiment of inducing models for a long string generated by a finite automaton. The advantage of using a finite automaton for parsing, is that it is a particular case of a finite memory model, hence we can extend a failing model m into a model m' by supplying a missing transition and we can also compute the code length that m' would have produced for the previously parsed characters. In short, we keep a count $c(q)$ for

how many times each state was used. When one transition is added to a state q , the code length can be updated by

$$\text{codelength}(m') \leftarrow \text{codelength}(m) + c(q)(\log(|A_q|+1) - \log(|A_q|))$$

The algorithm we used is similar to the one outlined for Finite Memory Models. It maintains a set H of hypotheses (A, c, g) where

A is an automaton with the transition function possibly not specified for every pair (state, input),

c is the current state, and

g is the current compression gain = $|\text{input}| - (|\text{output}| + \text{complexity})$.

It performs a 'beam' search on a space of all possible automata (given the input alphabet) by selecting all hypotheses that failed to predict the current symbol and expanding them in all possible ways. When the number of hypotheses exceeds a certain limit, the hypotheses that are less performant in terms of compression are discarded.- We applied this algorithm on a string generated by taking random decisions for the '+' operator in the following regular expression : $(aab(aab + abbbabbb)abbb)^*$

Algorithm 3.6.1

```
H <- {(A,q,0) | A has one state (q) and no transition specified }
Loop
  a <- next input
  For all (A,q,g) in H do
    If transition(q,a)=q'
      g' <- update(g)
      Replace (A,q,g) in H by (A,q',g')
    else ( if transition(q,a) is not specified)
      For all states s of A do
        Let A' be A with the new transition q,a -> s
        Add (A',s,g') to H
      Let A' be A with a new state s and a new transition q,a -> s
      Add (A',s,g') to H
  If |H| > Limit
    Discard from H all triplets (A,c,g) where  $g < \max(g) - g_0$ 
```

This is a summary of basic regularities found at each stage:

Symbols read	Models discovered	Complexity	Compressibility
24	$m_1: (aab+abbb)^*$	6	0.5
50	m_1	6	0.6
	$m_2: (aab+abbb(aab+abbb))^*$	12	0.5
241	m_1	6	0.69
	m_2	12	0.67
	$m_3: (aab(aab+abbbabbb)abbb)^*$	19	0.84

We would like to make a few remarks about this experiment:

(1) $L(m_1) = L(m_2)$ but m_2 is larger hence less efficient.

(2) $L(m_3)$ is included in $L(m_1)$ and m_3 is composed of two levels, each having the structure of m_1 :

If $x = aab$, $y = abbb$ then $m_1 = (aab + abbb)^*$, $m_2 = (xxy + xyxy)^*$.

Let us use \otimes to denote this composition of models. Therefore we can say that $m_3 = m_1 \otimes m_1$

(3) Any other models will have a lower compressibility factor than m_3 .

(4) If we were not constrained to the given class of models (finite automata), then $\text{plex}(m_3)$ would have been just a little higher than $\text{plex}(m_1)$ since m_3 is essentially m_1 plus a small control part that causes the recursion. However the smallest finite automaton that has the effect of $m_1 \otimes m_1$ is about three times larger. This complexity gap between m_1 and m_3 makes m_1 a basic regularity easily distinguishable.

(5) Let m be an automaton $[A, Q, \delta]$ that is able to parse a finite string s . Define $A_q = \{ a \in A \mid \delta(q, a) \text{ is defined} \}$. For every transition from a state q , m will have to produce a code of length $\log |A_q|$. Assuming that we know the probabilities $p(q_i)$ of each state being used in parsing and the probabilities $p(q_j \mid q_i)$ of transitions $q_i \rightarrow q_j$ being used, we can be more precise. Then the average code length for a symbol of s will be

$$H(A/m) = \sum_i p(q_i) (-\sum_j p(q_j|q_i) \log(p(q_j|q_i)))$$

the average information necessary to determine what symbol might follow a string, given that the string is parsed by the model.

Example 3.6.4 [Caianiello-71] is the earliest example known to us of stepwise induction on texts. He applies the Procrustes algorithm (example 3.6.1) to identify several levels of library models for a finite text. Once the first set of routines x_i is identified, a new alphabet is created by assigning a new symbol to each routine. Since the original text is a sequence of routines occurrences, it can be re-presented now as a new text over the new alphabet. This re-presentation of the original text is analyzed, in turn, by the same algorithm.

[Caianiello-89] continues these ideas and considers two types of routines: sequences of symbols and classes (clusters) of symbols. He shows that many syntactic categories of natural language (syllables, words, noun phrases etc.) can be identified this way and suggests this as a natural approach to learning the syntax of natural language.

Example 3.6.5 Let us look again at example 3.5.5. We established that a clustering is a library type model. In our case the set of points will be coded into a set S_0 of cluster centers c_i and several sets S_i of points x_{ij} each one having a center as origin. If the set S_0 or any of the sets S_i do not contain a uniform distribution of points, (which in *models* language is equivalent to non random codes), then clustering can be applied again, generating a two level hierarchy. If it is applied on S_0 it creates a second level above the first one, while if it applied on any of the sets S_i , the second level will be underneath.



While for finite structures like the one in examples 3.6.1 and 3.6.2 we could compute the compression factor at each level of complexity, and the basic regularities were easy to identify, infinite structures require an incremental approach. Therefore, at each step we have only an approximation of $\text{press}(m)$. A model that seems to be a local maximum for $\text{press}(m)$ may turn out not to be a basic regularity in view of further data. Research in conceptual clustering, for example, took this into account by allowing, in certain conditions, splitting and merging of previously identified clusters. (see the COBWEB and CLASSIT algorithms in [Gennari-89]).

Our algorithm 3.6.1 periodically discards low compressibility hypotheses. While missing transitions in a model will eventually cause its failure, complete models may

emerge for which some paths in the transition graph are no longer used. As a consequence, some code values are no longer used. The algorithm could be augmented to recognize this situation and it could eliminate the unused paths in the graphs, lowering the complexity of the models.

The stepwise approach to compression may become quite complex. Let us look at the general picture. Suppose that by some induction algorithm we found a model m_0 that parses a string s into a code string s_0 such that $|m_0| + |s_0| < |s|$, but more compact descriptions of s are possible. Two further compression paths are possible:

- (a) compress s_0 : find a model m_1 and a code string s_1 such that
 $s_0 = m_1(s_1)$ and $|m_1| + |s_1| < |s_0|$
- (b) compress m_0 : find a model m^1 and a code string s^1 such that
 $(\forall x)(m^1(s^1, x) = m_0(x))$ and $|m^1| + |s^1| < |m_0|$

While (a) is a further induction on strings (this time code strings), (b) depends on the availability of induction algorithms that work on descriptions of object of class M . As an example, let M be the set of finite automata, and the string s be a concatenation of context free expressions like $((()))()(((())())())()$. M will never capture the balance of parentheses. That will be implicit in the structure of m_0 but cannot be used by it to increase compression. However, a different compression algorithm may recognize similar parts in the transition graph of m_0 and code them as subroutines.

Moreover, the levels of a hierarchical structure might share a large amount of mutual information. A straight forward example would be a class of structures that accept models like $m \otimes m \otimes m \otimes m$ (where \otimes stands for composition). The sequence of levels may become an object for induction too. (see Figure 3.6.12)

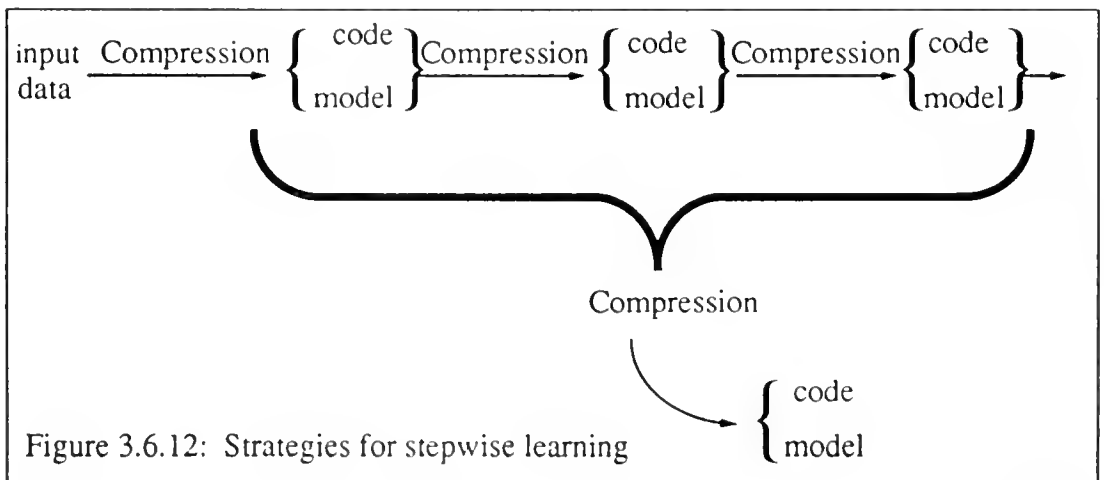


Figure 3.6.12: Strategies for stepwise learning

4. Learning an Environment by Explorations

In this chapter we focus on algorithms endowed with a body, that is, simulated agents that engage in active learning. They accept an infinite string produced by an unknown generator but they can control it, to a certain extent, by acting on the generator.

Induction is concerned here with the sensory-motor stream, that is, an alternation of actions and generated characters. From a psychological perspective these investigations can be seen as computational models of cognitive development. Their ultimate goal is to provide a basis for language learning, and for assessing the program's acquired knowledge through conversation instead of vivisection.

We study active learning in three, increasingly complex situations. First we consider an agent that feeds input symbols to, and reads the output of a Moore automaton. We present two incremental algorithms. One refines an initial model to maintain its determinism. The other accumulates partial regularities, and alternates between three modes: regularity recognition, prediction based on the recognized regularity, and confusion. Then we consider a world of loosely coupled automata where the FMM's, introduced in the previous chapter, provide a versatile way to capture regularities. We investigate regularities that correspond to various substructure homeomorphisms. We show that, under certain conditions, induction in this space operates a construction of reality, that is, it explains the emergence of both a space map and independent objects. Finally we consider an agent endowed with a complex eye. In this context we show how complex computations are derived from acquired statistical correlations.

4.1 A Psychological Perspective

While the previous chapter was concerned with induction of models for various symbolic constructs, we would like now to expand our understanding of learning to an activity of attaching meaning to symbolic constructs. So far the learner was, to a certain extent a passive box that we kept stuffing with symbols from an infinite string. We required that string to contain all relevant information that the learner was supposed to extract. We shall look now at learners that can act, and by that, influence the content of the string of symbols presented to them.

[Winston-80] provides a good starting point for our discussion. Here is a simplified account of it. The author takes the plots of several plays by Shakespeare and reduces them to a set of graphs. The names of the main characters label the vertices. Words for relationships between the characters or actions involving them label the edges of the graph. These graphs are fed to a program that finds common subgraphs and, consequently, establishes mappings between node labels and between edge labels. It is

claimed that the program found common meaning in those plays.

Now, we know that, for the program, those plays are as good as, say, a configuration of pebbles, and the meaning is nothing more than counting how many blue pebbles touch red pebbles. We may wonder how were those labels chosen, and whether the common subgraph wasn't put there to start with. The domain is strongly affected by subjectivity. Our intention is not only to create a transposition of the chinese room argument [Searle-80], but to make obvious a mixture of two fairly distinct lines of research in AI.

On one hand there is the struggle to create adequate languages, in fact full formal systems, that are able to represent, by their expressions and computations on expressions, the kind of connections that people make from one sentence to another when they *reason*. A well designed system could be used as a language to spell out situations, actions, and natural constraints among them. The *meaning* still resides in an interpretation that is essentially exterior to the system.

On the other hand there are the attempts to create a formal system that can represent the emergence of meaning. Such systems should represent both model agents and a model world that is to acquire a *meaning*, for those agents, through their interaction. They should offer computational equivalents for some intrinsically psychological concepts.

Using such a complex universe as a Shakespeare play seems misleading in both situations. In this chapter we show that simple toy worlds can still be useful laboratories. Working on geometrical and mechanical concepts allows for clear definitions, and yet, ample complexity.

4.1.1 A Semantic Basis for Learning Language

In section 3.1 we argued that a passive learning algorithm can only learn the syntactic constraints of the data presented to it. A complete representation involves an interpretation, that is, a mapping between a language structure and a represented domain structure. Therefore, learning a representation requires access to the represented objects, which may reside outside the realm of the symbolic constructs.

A similar reasoning appears in natural language learning theories. For example, [Fodor-75] uses this argument to postulate a radical language of thought hypothesis. He starts from the assumption that learning a language is a special case of hypothesis formation and confirmation. A speaker of A that learns the language B, will have to create and confirm hypotheses of the type

$P(x)$ is true in B \Leftrightarrow $Q(y)$ is true in A

Given the left hand side of this equivalence, the learner must be able to create the right hand side. Therefore, in order to explain how any language is assimilated, one must assume an existing language in which the new one is learned. From here, Fodor concludes the need to postulate an initial, innate language, that we shall refer to as *mentalese*, necessary to learn the first natural language. The expressive power of mentalese must at least equate the expressiveness of the language being learned.

The idea of an innate linguistic nucleus was also promoted by Chomsky [Chomsky-79] for different reasons. Among his arguments is the observation that a man can master the grammar of a language although, throughout his life, he encounters the vast majority of sentences only once.

An alternative view of cognitive development results from the research of the child psychologist J. Piaget who thinks that mentalese is not innate but acquired as a prelinguistic intelligence [Piaget-54,69,79]. The two positions were extensively debated [Piatelli-Palmariny-79],[Powers-89]. Still, heredity plays its part in directing learning and modelling behavior [Gould-87]. Biologists and psychologists have yet to strike some middle ground to explain the cooperation of innate mechanisms and interaction driven construction in learning.

Piaget's view suggests a construction of conceptual structures through interaction with the environment. This is what makes it especially appealing for computational modeling. He argues that no knowledge is based on perception alone since perception always occurs embedded in an action scheme. Piaget's theory of cognitive development in infants is centered around a set of skills that he calls sensory-motor intelligence (SMI), and that precedes and supports linguistic intelligence. He stresses the importance of active interpretation and organization of experience.

The main element of the theory is the *action scheme*. Reflexes are initial action schemes. The infant grasping an object that touches her hand, or sucking a nipple that touches her mouth are examples of reflexes. Schemes evolve through variations introduced by repeated application. Piaget describes several stages of this evolution:

New objects - in fact sensorial information about them - that happen to trigger a reflex, are assimilated into it.

Some modified reflexes are no longer fired by a trigger but used repetitively and changed into habits. For example, the infant learns to suck a thumb and makes a habit out of it.

Mindless exercising the reflexes leads to discovery of predictable correlations

between premises and results, thus, to creation of goals.

As objects are assimilated into a schema, it becomes more elaborate. Unexpected results are linked to triggering images. One of the skills achieved at this stage is the coordination of hand and eye movements.

The child learns to grasp different objects, then move her hand towards remote objects in her visual field.

Observed external processes are internalized into purposeful activity. The infant seizes an adult hand and moves it towards unreachable objects. New means are searched. The infant pulls a blanket on which a remote object lays.

Certain invariant images are recognized and memorized even when they temporarily disappear from the visual field. Objects are thus represented as permanent entities. They become elements for what Piaget calls the construction of reality.

Certainly, the above progression can offer nothing more than metaphorical guidance, since its vocabulary is not directly translatable into computational equivalents. Still, from the point of view of this paper, the interest lies in offering a corresponding mechanism for this evolution and adaptation of actions to the structure of the environment. A model for the acquisition of SMI might explain how mentalese - at least a computational version of it - is constructed.

Example 4.1.1 [Drescher-86] offers an interesting model for the initial stages of SMI development. His agent is endowed with a hand and a square, tiled retina. Both hand and retina can be moved. When visible, the hand occupies one retinal tile. The cognitive evolution is modeled as a process of generation and pruning of action-schemes that eventually achieves an eye-hand coordination and a set of plans to reach visible objects. His schemes are structured like rules:

```
premises -----> conclusions
              action
```

Pruning of rules is based on accumulated statistics of performance. The generative methods are listed here.

Initially, premises and conclusions are empty lists. Actions are taken at random. Sensorial stimuli that happen to precede or follow the action are incorporated into premises and conclusions, respectively, generating new rules. Validity statistics are accumulated. Valid rules are chained together into new ones that represent sequences of ac-

tions:

premises -----> conclusions
 sequence of actions

Once valid rules are found that reach a conclusion from any premise, that conclusion is promoted among actions.



For a general approach, we need to link this kind of psychological modeling to the inductive vocabulary developed in the previous chapter. In a computational model of SMI development, schemes will correspond to induction hypotheses, that is, to predictable areas of the universe. In order to be more precise about schemes and their properties, we have to discuss them in a formal context. The next section provides this context.

4.1.2 The Agent Living in a Passive, Deterministic Environment

We think that modelling simple situations might be a clarifying and fertile activity. The simplest model should contain one agent and a passive universe that changes its state only as a result of an agent's action, and always in a predictable way. We can easily draw analogies between this problem and several *natural* situations: a robot moving parcels among a set of rooms or exploring a deserted territory, but also a child playing by herself in a room full of toys. We shall give a general statement of the problem and then, in the remainder of this chapter, we shall look at further simplifications of it that allow actual solutions.

A few details on the possible anatomy of the agent are needed here. As a first principle, the agent will have no direct access to a representation of the universe that it inhabits. It will only be endowed with senses and actions. It will be able to execute only one action at a time. The string of these actions alternated with the images provided by the senses will be its description of the universe. We shall call such a string a *sensory-motor stream*.

If our model agent is endowed with a body, then some actions will have effect on the body and part of the senses should inform the agent about it. Let us call those the internal senses. The body may have its own autonomous mechanisms generating needs and informing on their fulfillment. The body, if it exists, is, from a cognitive point of view, part of the unknown universe. The agent will have to discover its reactions and learn to coordinate its movements. The presence of a body would offer a general goal for the learning mechanism. The agent would have to perform an optimization in terms

of maintaining the values of the internal senses within certain prescribed limits, i.e. *keep the body happy*.

A simpler anatomy may involve a body with no needs. The senses would only provide information on the universe itself and the position of the body. The purpose of learning in this case would be purely predictive. Given the current indications of the senses and a memory of previous experience, what will be the result of a certain action. In this chapter we shall settle to this simpler version.

Here is a more formal statement of the problem. An agent A is endowed with a set M of moves that it can apply on its environment and a set of sensors that give it an image \mathbf{e} of the environment. The structure of \mathbf{e} depends on the particularities of the sensors but we can assume that each sensor is contributing a number of bits of information, therefore we can take \mathbf{e} to be a vector of bits. Since the environment is passive, \mathbf{e} changes only as a result of applying a move from M . Let E be the set of all possible values for \mathbf{e} . The way we defined \mathbf{e} , E is finite. Each move $\mathbf{m} \in M$ is a mapping $\mathbf{m} : E \rightarrow E$ that causes a transition in E . The entire universe of interactions between A and its environment can be represented as a finite automaton U with states E and alphabet M .

It may seem from the above considerations that all A has to do, to know its environment, is to learn the transition functions $\mathbf{m} : E \rightarrow E$, for all $\mathbf{m} \in M$. A simple-minded algorithm

```
Loop
  Pick a move
  Apply it
  Memorize the transition
```

serves this purpose, but suffers from obvious shortcomings:

(a) Interesting instances of E are enormous and memorizing all transitions is impractical if not impossible.

(b) The sensors may present to A only a partial *view* of the environment. In this case there is a set Q of hidden states for which E is only a projection. That is, there exists a non injective function $f : Q \rightarrow E$ that must be discovered together with the real transition functions $\mathbf{m} : Q \rightarrow Q$ only by observing the projections $f(q), \mathbf{m} \mapsto f(q\mathbf{m})$ that are no longer deterministic. The case where \mathbf{e} is a scalar will be investigated in the next section. The set of transitions here is even larger, and memorizing them might be, again, impossible.

(c) Most important, regularities exhibited by sequences of transitions (algebraic properties of moves composition, macrooperators that achieve an effect

independent of state, etc.) should be captured by an adequate representation of the universe, one able to support purposeful activity and communication.

Therefore we need to capture regularities of the real transition function and organize them in a concise representation. That is why we propose to consider SMI schemes equivalent to regularities of the sensory-motor (SM) stream. Thus, *the learning process for A becomes a problem of induction on the SM stream.*

Schemes must eventually reach a certain expressive power. As programs, schemes should lend themselves to refinement and composition in order to attain the required complexity. So, in this computational context, the contradiction between the nativist and constructivist theories of cognitive development seems to vanish. Indeed, *mentalese is here the computational class of our induction hypotheses.* It may range from any simple class, like regular languages, up to the set of partial recursive functions.

The difficulty remains in explaining how appropriately complex concepts could emerge. As expressions for partial knowledge, schemes will evolve to incorporate new situations. This will be our equivalent to assimilation.

Schemes should become elements for an ontology of E. They should identify those components of the universe's structure that communicating agents may have to speak about.

[Russell-89] discusses succinct representations that provide a good enough basis for achieving other objectives, like planning sequences of actions to attain *good* states (for ex. finding food) or avoiding *bad* states (being trapped, bumping into obstacles). Here are two of them:

- (i) $\text{condition_1}(\text{state}) \Rightarrow \text{condition_2}(\text{result}(\text{state}, \text{action}))$
- (ii) $\text{condition_3}(\text{state}) \Rightarrow \text{result}(\text{state}, \text{action}) = f(\text{state})$

In other words,

- (i) predicts that some condition will be satisfied by the resulting state if a certain action is applied in a certain situation.
- (ii) describes a way to explicitly build the resulting state.

Note that induction of such rules is constructive, since it requires creation of representations for new predicates (*condition_1*, *condition_2*, *condition_3*) and functions (*f*). Also, *state* itself is a constructed entity. All an agent knows consists of sequences of actions and sensory images. A state will be represented by a perception of how it was reached or by its effect on further perceptions. (This will be made clearer in the

next section. For instance, the predicate *condition_1* may be computed by the SM sequence

look-forward, see-opening, extend-left arm, feel-wall-contact,
extend-right-arm, feel-wall-contact

The *state* in (i) and (ii) above is an element of *Q*, that is, an atomic state of the entire agent / environment system. A perception of the separation of the agent from its environment or of independently moving objects in the environment involves a construction in which each component is denoted by a separate token.

Also, the predicate *condition(state)* denotes possibly many states of the universe. It is a representation for an agent centered situation. It may mean something like *The narrow street is in front of me*. They are simpler, since, if there are several *narrow streets*, determining exactly which one it is may require a more complex predicate. Therefore, during the accumulation of knowledge, *such constructs will probably occur before more precise ones*.

[Chapman-87] argued for a cognitive model that learns such simpler constructs. Their correspondents in natural language are *indexicals*. Planning of purposeful sequence of actions, even if prone to errors, is computationally simpler if based on such constructs. For our purpose they also offer a flexible representation for valid partial knowledge. To continue our example, an instance of (i) may be the following *partial regularity*:

The-narrow-street-is-in-front-of-me (state) =>
I-am-stuck (result (state, advance))

In the following sections we shall illustrate in details some instances of evolution of schemata and construction of adequate representations.

4.1.3 Some Future Themes

We would like to stress here the importance of this type of active learning problems by suggesting some generic goals.

(i) How do we know that our simulated agent learned something about the universe? By inspecting the internal representations it created during exploration. If they are in some way isomorphic to parts of the environment, we shall consider that those parts were discovered and learned by it. To assess the general learning performance, we can simply use the frequency of correct prediction of action results.

(ii) Can we assess the agent's performance without *looking* inside it? Assume an agent endowed with a small set of goals (needs) : avoid bad things, look for good things. During the simulation, we feed to it, besides the sensory image, a binary signal for *good* and *bad*. The goal of learning will be to maximize the frequency of *good* signals. Assessment becomes a matter of exterior inspection of behavior.

In the above situation we force the agent to learn a binary classification. The environment being deterministic, the agent will learn, for example, that the perception of a *blue triangle* is bad for *advances* but good for *turns*. We can also say that we are providing it with a one bit description of its state. We *are talking* to it in the simplest language of positive and negative reinforcement.

(iii) Imagine now the same agent also equipped with a special sensor, say *the ear*, the values of which are character strings. We use it to provide short English imperative sentences like "Avoid the blue triangle", "Reach the blue triangle". We use the good/bad signal to reward execution of those orders. We evaluate learning performance by the success of conditioning. An agent reacting properly to the above sentences, must have assimilated, among other things, the concept of blue triangle.

(iv) Finally, add to the anatomy of our agent, *a mouth*, that is, add the letters of the alphabet to the set of actions. Let the agent inhabit a universe together with some deterministic robots that obey orders and provide advice. In addition of its descriptive function, *language* can now be learned as a mean for influencing other agents, that is as *a universe of elaborate actions*. Agents may acquire different representations for the same situation but the constraints of communication will force them to be valid.

4.2 Model World 1 (Induction of Moore automata)

Let us concentrate on the simplest situation: an agent with a scalar sensory input. Therefore, imagine an agent that can choose from a finite set of actions, one at a time, say it can push one of a set of buttons available to it. The only thing that it can perceive is a colored light. Sometimes, as a result of executing an action (pushing a button), the color of the perceived light can change. There are only a finite number of colors, known in advance. As an equivalent problem, we may imagine a robot that wanders through a maze. At each point it can execute a one of a finite set of movements (advance, turn left, turn right, etc.) The robot has a very simple eye focussed on the wall in front of it. The eye conveys the color of that wall.

In this case the entire environment can be modeled as Moore automaton A . At any moment the agent is associated to one of the states and its eye perceives the output of that state. Execution of an action will cause a transition of A and the state may change. The output (color) of the new state may either be different from that of the previous state, or not.

This situation was investigated in [Botta-90]. We shall present here the main results without going into the details of the algorithms since they are given in the referenced paper. We shall use the word 'color' for the output of a state q .

Given a finite set of colors K , a finite set of states Q , a finite input alphabet Σ (the symbols in Σ represent the agent's actions), an output function $\phi: Q \rightarrow K$, and a transition function $\delta: Q \times \Sigma \rightarrow Q$, a Moore automaton is the quintuple $A = [\Sigma, Q, K, \delta, \phi]$. We assume that A is deterministic, that is δ is a proper function. However, since two different states might have the same color (i.e. $\phi(q_1) = \phi(q_2)$), the agent may confuse them at first but should eventually be able to distinguish them. The necessary knowledge can only be acquired by executing actions (hence transitions). Therefore we have to add these assumptions:

- (a) For every pair of different states, there must be a sequence of transitions that applied to both will result in states of different colors. If two states do not fulfill this condition they will always be perceived as one. That is why the automaton induced by the agent can only be a minimal model for the unknown one.
- (b) The transition graph of the automaton must be strongly connected since the agent may be able to reach any part of it several times.

[Botta-90] presents two approaches that are summarized below.

4.2.1 Characterization of states by sets of tests

According to assumption (a), for two distinct states of the same color there must be an input sequence that applied to those states would cause the automaton to behave differently, therefore helping to distinguish the two states. We introduce here several terms that will express the idea more clearly. Note that we use $q a \rightarrow q'$ as a shorthand notation for $\delta(q,a) = q'$. We also extend this to sequences : $q a \rightarrow q'$ for a in Σ^* .

Definition 4.2.1 A test is a sequence from $T = K(\Sigma K)^*$.

Definition 4.2.2 Passing a test is defined recursively :

A state q passes a test $t = k$ iff $\phi(q) = k$.

A state q passes a test $t = k a t'$ iff $\phi(q) = k$ and $q a \rightarrow q'$ and q' passes t' .

Restating what we said above, if q_1 and q_2 are distinct, then there exist a test t such that q_1 passes t and q_2 doesn't. In fact every state can be characterized by a set of tests that distinguish it from all other states. Since, in distinguishing two states, it is useless to continue testing after the first input symbol that caused distinct behavior, we will assume that the tests that characterize a state are minimal. That is a test $t = t' a k$ (t' test, $a \in \Sigma$, $k \in K$), that distinguishes two states from each other, is minimal if t' doesn't distinguish them.

Definition 4.2.3 $\text{states}(t) = \{ q \in Q \mid q \text{ passes } t \}$

Definition 4.2.4 $Q_k = \{ q \in Q \mid \phi(q) = k \}$

Definition 4.2.5 A state q fails a test t because $t = t_0 a k b$ for some $t_0 \in T$, $a \in \Sigma$, $k \in K$, and $b \in (\Sigma K)^*$ and q passes t_0 , $\delta(q, t_0 a) = q_0$, but $\phi(q_0) = k' \neq k$. This failure defines a new test $t' = t_0 a k'$ such that q passes t' . We shall denote t' as $\text{correction}(t,q)$.

Definition 4.2.6 Sets of tests can be represented by test-trees, that is, trees with nodes labeled with elements of $K \cup \Sigma$. The root label and the leaf labels are colors. All children of color labeled nodes are labeled with input symbols. All children of input symbols labeled nodes are labeled with colors. With these constraints we make sure that the sequence of labels on any path from root to a leaf is a valid test. We shall also label the leaves with symbols from Q to associate the tests determined by the leaves with the states that passes them.

Definition 4.2.7 Note that all tests necessary to identify the states of the same color k can be represented by a test-tree whose root label is k . We denote by $t_{\text{set}}(k)$ a minimal set of tests that distinguish among all states of color k .

Definition 4.2.8 Similarly, $t_set(q)$, is a minimal set of tests that identify a state q . This is a *single state* test set. It can be represented by a test-tree in which color labeled nodes are single children. This constraint reflects the fact that the automaton is deterministic and a transition caused by any sequence of input symbols will result in only one new state, hence one color.

Definition 4.2.9 A *single sequence* test set is represented by a test-tree where each color labeled node has at most one child. *The tests in such a set can be run by feeding the automaton a single sequence of input symbols:* One starts with the symbol that labels the child of the root. From the grandchildren of the root one chooses the one labeled by the color currently observed. if there is no such grandchild, all tests fail. Otherwise one continues the same way.

We shall use the concepts of test and test sets in the following way. We will have the learner assume initially that *there is only one state for each color in K* . In the course of its exploration the learner may find a symbol a that, when applied to a state q of color k , may cause transitions to two distinct new states of colors k' and k'' . In that case the learner will replace the state q of its model by a pair of new states q' and q'' characterized respectively by the tests $k \ a \ k'$ and $k \ a \ k''$. Let us sketch now the algorithm more precisely. (The question mark is used as a wild-card character. In lines 5 and 12 it stands for any resulting state, in line 13 it stands for any originating state and any input symbol.)

Algorithm 4.2.1

```
1 Initialize the current state  $q$  to a one node test tree labeled with the
  currently observer color  $k$ 
2 Loop
3   Apply an action symbol  $a$  and let the newly observed color be  $k'$ 
4   Run the tests in  $t\_set(k')$  in order to identify the new current state  $q'$ 
5   If no transition  $q \ a \rightarrow ?$  exists
6     Record  $q \ a \rightarrow q'$  as a new transition
7   If there is a transition  $q \ a \rightarrow q'$ 
8     Continue the loop
9   If there is a transition  $q \ a \rightarrow q''$  ( $q'' \neq q'$ )
10    Split  $q$  into  $q$  and  $q_1$ 
11    Record the new transition  $q_1 \ a \rightarrow q''$ 
12    Delete all other transitions out of  $q$  ( $q \ x \rightarrow ?$ ,  $x \neq a$ )
```

Note that, in line 4, all the tests in $t_set(k')$ must be run in order to identify the new current state. In a trivial case, the tree representing $t_set(k')$ is a one node tree. Then, there is nothing to identify: there is only one state colored k' in our model. Otherwise, we have the problem that running a test entails leaving the state q' .

If $t_set(k')$ is a *single sequence set*, all its tests can be run by feeding one sequence of input symbols. This is certainly possible but, by the time we know the results of the sequence, the action in state q' has already been decided, namely the first symbol of the test sequence. Therefore each time the learner sees the color k' it has to apply the same input, the one dictated by the test and will never be able to know what would the automaton do if fed another input.

If the tree is a multiple sequence set, the situation is worse because after running one sequence the learner has no way to ensure a return to q' to run the others. Several simplifying assumptions can be considered in order to continue from this impasse:

- (i) The learner is assisted by a teacher that “knows” the automaton and can supply, on demand, the results to any test without running it. Since tests are expected properties of the current state, this procedure could be amenable to the framework presented in [Angluin-78].
- (ii) There is a simple way for the learner to force the automaton back into the state q' . For example, assume that for every symbol a in S there is an inverse $inv(a)$ that causes an inverse transition:

$$\delta(q_1, a) = q_2 \iff \delta(q_2, inv(a)) = q_1$$

This is possible only if all $\delta(_, a) : Q \rightarrow Q$ for a in Σ are bijective. (Permutation environment).

- (iii) After a *long enough* experiment while the learner's model is not contradicted by the automaton's behavior, the learner commits to its knowledge, i.e. assumes that the predictions of its model are correct. Then the model can be used to find a way back to q' . The problem with this approach is that there is no 'long enough' experiment, if the number of states is unknown. For any finite experience accumulated by the learner, an 'adversary' automaton can be found that complies with that experience but then causes a surprise that invalidates the model built so far. In this case commitment hampers further refinement and model building must restart from the beginning.

A few details about line 10. Assume that we have a recorded transition $qa \rightarrow q'$ but, when applying a in state q' , the resulting state is q'' . In order to keep the model deterministic, q must be split into q and q_1 such that $qa \rightarrow q'$ and $q_1 a \rightarrow q''$. Also let k be the color of q and q_1 . This split is achieved by creating a test set for q_1 and, in some cases, by refining the test set of q , in order to keep q and q_1 distinct. Note that $t_set(q)$, $t_set(q')$ and $t_set(q'')$ are known at this point. One starts from the tests in

$t_set(q)$. The tests of the form $k \times \dots$ with $x \neq a$ will be inherited by both q and q_1 . On the other hand, the tests $k a \dots$ will make a difference. The details are given in the referenced paper.

4.2.2 Characterization of states by id_sequences

A second approach discussed in [Botta-90] is to use a sequence of previously seen colors, instead of the current state, to predict the current state. To be more precise, let us introduce the following definitions :

Definition 4.2.11 A test t is an *id_sequence* for a state q iff for every state q' that passes t , running t on q' will bring the automaton into the state q . Therefore, even if we don't know q' we can run the *id_sequence* and, if we succeed, we are sure to be in the state q .

Definition 4.2.12 A subset R of Q filtered through a test t is another subset of Q , denoted by $R|t$ and defined as $\{ q \mid (\exists q' \in R) (q' \text{ passes } t \text{ and } q't \rightarrow q) \}$

This follows the intuition that it is possible for a sequence of colors and actions previously encountered to uniquely determine the current state or at least the reaction of the automaton to the next input symbol. In fact we can prove the following:

Proposition 4.2.1: For every color k in K , there is a state q in Q such that there exists an *id_sequence* for q starting with color k .

Proof: Let us consider $Q(k) = \{ q \in Q \mid f(q) = k \}$. If $Q(k)$ is a singleton $\{ q_0 \}$ then k is an *id_sequence* for q_0 . If $Q(k)$ has at least two states q', q'' , there must be a test t_0 that discriminates q' from q'' . Say q' passes t_0 and q'' doesn't. It follows that $Q_1 = Q(k) | t_0$ is not empty, since it contains q_1' where $q't_0 \rightarrow q_1'$, but it has a smaller number of elements than $Q(k)$ since q'' has no correspondent in it. Applying the same reasoning again, either Q_1 is a singleton $\{ q_1' \}$ and t_0 is an *id_sequence* for q_1' , or there must be a test t_1 that discriminates between two states q_1' and q_1'' in Q_1 . Since $Q(k)$ is finite and every step reduces the number of elements without producing an empty set, eventually there must be a test $t = t_0 t_1 \dots t_n$ such that $Q(k) | t = \{ q_{n+1} \}$ and t will be an *id_sequence* for q_{n+1} .

Proposition 4.2.2: For every state q in Q and color k in K , there is an *id_sequence* for q starting with color k .

Proof: Given a color k there exists a state q_0 and a test t_0 starting with k such

that t_0 is an id_sequence for q_0 . (from proposition 4.2.1) But every state q is accessible from that q_0 (remember the strong connectivity assumption) therefore for every state q there must be a test $t_{0,q}$ such that $\{q_0\} \vdash t_{0,q} = \{q\}$. Then $t_{k,q} = t_0 t_{0,q}$ is an id_sequence for q that starts with the color k .

Based on these two propositions, we present an algorithm that remembers all finite behavioral sequences of length n and evaluates their predictive capabilities. The model built this way will not be a complete model of the unknown automaton. Sometimes the learner will be able to make a prediction, other times it will be confused. When expectations are contradicted n will be increased in an attempt to remember longer past sequences.

It is convenient to hold behavioral sequences as test trees. We store all encountered sequences of $K \times (\Sigma \times K)^*$ up to some finite length $2n+1$, that is n steps, and consider them candidate id_sequences. The prediction capability of these candidate is to be tested for another n steps. A candidate whose tree of continuations is not a single state tree is eliminated. In the algorithm this is done by marking the color nodes of that candidate as "multistate". If all candidates of length $2n+1$ are eliminated, all id_sequences must be longer, therefore n is increased and the search continues.

Algorithm 4.2.2

```

T <- a set of one vertex test trees each vertex labeled with a color from K
C <- { v } where v is the vertex labeled k and k is the color currently seen
Loop
  Apply some random action a from  $\Sigma$  and let the new color be k'
  For all vertices v in C do
    If the depth of v in its tree is  $4n$ , Remove v from C
    If v has no child labeled a
      Create v' child of v with label(v') = a
      Create v'' child of v' with label(v'') = k'
      Replace v in C by v''
    else ( v has a child v' labeled a )
      if v' has a child v'' labeled k'
        Replace v in C by v''
      else ( v' has no child labeled k', but it certainly has other children )
        Create v'' child of v' with label(v'') = k'
        Mark v and all its ascendants as "multistate"
        Replace v in C by v''
  If all nodes at depth  $2n$  are marked "multistate", increase n

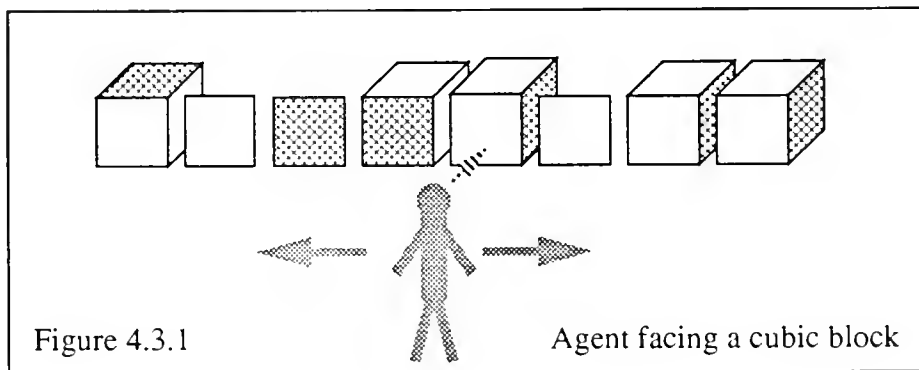
```

This algorithm will accumulate partial models. Each one consists of an `id_sequences`, that, once encountered, predicts the resulting color. A model whose prediction fails is refined by increasing the sequence.

4.3 Model World 2 (Objects and Simple Eye)

Example 4.3.1 Let us imagine an agent, represented as the stick figure in figure 4.3.1, exploring an unknown space. The space contains several kinds of objects. In the drawing above the objects are blocks and flat squares. Each face of these objects is painted with one of three colors, say A,B and C. The squares are painted with A and B. The blocks have opposite faces of the same color and adjacent faces of different colors. The agent can perceive one face of one object at a time. It explores this space by executing at each moment one action chosen from a given set. For example: Actions = { L, R, V, H } where L = move to the left, R = move to the right, V = rotate vertically the object in front of the agent, H = rotate horizontally the object in front of the agent.

Such a domain can be represented by a Moore automaton. It can easily be made arbitrarily complex by increasing the number of objects, allowing them to overlap each other and allowing the agent to shift them, making the L and R moves dependent on some cyclic time schedule, etc. This results in an automaton with a very large number of states and the difficulty of learning this space (which is equivalent to inducing the automaton) grows accordingly.



This is an interesting domain because it can model many natural situations involving spatial explorations. Although the total structure of this space seems complex, it is not really prohibitive for learning because it is constructed from smaller, recurrent substructures that behave independently most of the time. It is suited to a stepwise induction approach because these substructures, corresponding to objects and fragments of space, can be learned before attempting a complete model. Even if a substructure is unique, it could be learned if it is encountered repeatedly by the agent during exploration and it behaves consistently for each encounter.

4.3.1 The class of domains.

Here is an attempt to describe an entire class of domains that involve spatial re-

relationships and simple interactions between objects. They include the following:

(a) SPACE. Both objects and agent are placed in a space that can be perceived as a collection of discrete, indivisible places or positions. An automaton can describe the space by representing the possible transitions between places.

(b) OBJECTS. Objects are described by simple Moore automata. The output (color) of the current state represents the currently visible aspect of the object. An object has an input alphabet that defines which actions have effect on it. As the result of an action the object will either execute a transition or will change its place.

(c) AGENT. The agent is represented as a collection of actions that are accepted as inputs by various automata. It can execute any sequence of actions. If the agent shares a place with an object some of its actions become inputs for that object. Also the visible aspect of that object is the only thing that the agent sees. Some other actions are acting on the agent itself by changing its position.

(d) INTERACTIONS. Most of the time direct actions from the agent will provide input for objects. However, depending on relative position in space, their states and other factors, objects can interact. If the conditions for interaction between two objects, say A and B, are fulfilled, then an action *a* applied to A can have one of the following effects:

- | | |
|------------------|-----------------------------------------------------------------|
| d.1 Propagation: | Another action <i>a'</i> is induced upon the object B |
| d.2 Blocking: | Action <i>a</i> has no effect on A |
| d.3 Bouncing: | Action <i>a</i> is <i>read</i> by A as another action <i>a'</i> |

(e) COMPOSITION. The interaction of two objects may take the form of composition, the objects being assembled into a larger one. The automaton representation for this one will be obtained in a predictable way from the automata of the two components.

Since the agent sees only one aspect at a time, and the number of possible situations is finite an entire such universe can be represented as a Moore automaton (M). That perceived aspect will be the output of the current state and we shall call it the color of that state. In order to build a representation of an universe the agent performs tests, that is, it applies a sequence of actions while checking the colors of the states it goes through (see Definition 4.2.1).

Proposition 4.3.1 Finite memory models can support the development of an adequate representation for this class of domains.

Argument. Each domain component (object, space, agent) has a finite number

of states, hence the entire domain is a finite state machine. Most domain components are loosely related therefore they are mostly independent, that is, most of the time an action will affect only one or two components. An FMM representation could allocate a small fragment of the WORK tape for each component state. This would allow a separate description of each component through transition schemata that affect only the appropriate fragment. Interaction among components could be represented by schemata that affect several fragments.

Assume that m schemata of the latter type are needed to describe all interactions. A complete description of the domain could be achieved with $N = m + \sum n_i$ schemata, where n_i is the number of schemata necessary for the i^{th} component. But, as long as m is small (that is, the objects are loosely coupled), N is substantially smaller than $\prod n_i$, that is, the number of transitions in a Moore automaton model.

Along these lines we can also argue that as soon as a component can be decomposed into two subcomponents sufficiently large and sufficiently independent, the FFM that makes those subcomponents explicit will have a complexity substantially smaller than the one that doesn't distinguish them. From all these we conclude that a model that makes all essential components explicit will be induced first by an algorithm that proceeds in order of increasing complexity.

Example 4.3.2 : *Noninteracting, nonmoving objects.* There is an object in every place. Objects cannot change places, only the agent can. Objects do not interact, that is, the agent can act on the object he shares the place with, but the effect of an action does not propagate to other objects. In this case $\text{Actions} = O \cup M$, where actions in O have effect on objects and actions in M (moves) change the place of the agent. A state of this universe has to contain the place of the agent (i), that also determines the current object, and the state of each object (q_i). A finite memory model work tape for it would be $[i \ q_1 \ \dots \ q_k]$, and these the necessary transition schemata ($m \in M, o \in O$). Note that the first line corresponds to a description of space and each line below it corresponds to an object. (X_i 's are variables)

$i, m \rightarrow i'$
 $[1 \ q_1 \ X_2 \ \dots \ X_k], o \rightarrow [1 \ q_1' \ X_2 \ \dots \ X_k]$
 $[2 \ X_1 \ q_2 \ \dots \ X_k], o \rightarrow [2 \ X_1 \ q_2' \ \dots \ X_k]$
 \dots
 $[k \ X_1 \ X_2 \ \dots \ q_k], o \rightarrow [k \ X_1 \ X_2 \ \dots \ q_k']$

Example 4.3.3 : *Agent with orientation.* Actions are *move* and *turn*. The agent has an orientation that determines the result of *move* and can be changed by *turn*. Each place contains a single state object. This way each place has a unique color. A state of this universe can be represented by the pair orientation-position [i p]. Transition schemata are:

i, turn -> i'
[1 p], move -> [1 p']
[2 p], move -> [2 p']
...

Example 4.3.4 : *Movable Object.* Actions are *move* and *push*. There is one object. The agent can *move* to a different place, or it can *push* the object to another place if it shares the place with it. A state will encode the place of the agent and the place of the object [a o]. Transition schemata are:

[p X], move -> [p' X]
[p p], push -> [p' p']

4.3.2 Objects as Equivalence Classes of Substructures

A *library* type model determines an equivalence relation among the fragments of a structure: *two fragments are equivalent if they are coded by the same routine*. The inverse is true in some cases: *an equivalence relation R is a regularity if its equivalence classes can be used to construct a compressed representation*. Here are a couple of equivalences that are applicable to our class of domains.

(a) Subgraphs.

Let us look at domains that contain a few types of objects but many objects of the same type. As long as the agent keeps dealing with one object, it has no way to distinguish it from another one of the same type. Only sequences of actions that cause it to move among places may enable it to create different representations for objects of the same type. The transition graph of the Moore automaton model of this domain will contain isomorphic subgraphs corresponding to objects of the same type. Here are the formal definitions. Let $A = [Q, \Sigma, K, \delta, \phi]$ be a Moore automaton with transition function δ and output function ϕ . The quintuple A has the same meaning as the one used in section 4.2. The notation $[q, k, a \rightarrow q' a']$ will be an abbreviation for $\delta(q, a) = q'$, $\phi(q) = k$, $\phi(q') = k'$. We define an equivalence relation on sets of states in Q :

Definition 4.3.1 $(\forall Q_1, Q_2 \subset Q), Q_1 \equiv Q_2$ iff

(i) Q_1 and Q_2 are disjoint

- (ii) The transition subgraphs determined by Q_1 and Q_2 are strongly connected
- (iii) $(\exists h \text{ bijective: } Q_1 \rightarrow Q_2)$ such that
$$(\forall q, q' \in Q_1) [q, k, a \rightarrow q', k'] \Rightarrow [h(q), k, a \rightarrow h(q'), k']$$

Definition 4.3.2 $(\forall Q_1, Q_2 \subset Q)$, $Q_1 \sim Q_2$ iff $Q_1 \equiv Q_2$ and they are maximal with respect to h , that is, there is no q_1 not in Q_1 and q_2 not in Q_2 so that $Q_1 \cup \{q_1\} \equiv Q_2 \cup \{q_2\}$

Assume that the current state is in a set Q_1 and there is another set Q_2 such that $Q_1 \sim Q_2$. As long as the transitions keep A in a state in Q_1 , the agent has no way to know whether it is in Q_1 or Q_2 . In figure 4.3.1 for example, two cubic blocks look identical as long as the agent only rotates them around their axes but doesn't move to a different object.

In the special case where the entire set Q can be partitioned in such subsets, a more compact representation is available for the initial automaton. A state q is represented by the pair $\langle u, v \rangle$ where u is the subset to which q belongs and v identifies it within that subset. A transition that keeps A within the same subset will affect only the v component.

Note that such a representation is a finite memory model (section 3.4) with a work tape length of 2. Moving from the former representation to the latter entails the emergence of separate models (concepts) for types of objects or for collection of spaces. Also, objects are basic regularities of this universe. They enable a stepwise approach to learning the domain.

(b) Tests.

Let us concentrate now on domains where there are many ways to achieve movement among places. An example is an universe of square tiles. The agent stands on a tile and has an orientation. It can only see the object placed on the adjacent tile, in front of it. The agent can advance (action: a) and turn around 90 degrees to the left (action: l). Also, the tiled surface wraps around in both directions after k tiles. Some regularities of this domain are:

- (1) the actions l^4 and a^k are no-ops
- (2) performing the action aal will always result in the agent facing the same tile as performing $lallla$

In a Moore automaton representation every place will be repeatedly represented for each of the four possible orientations of the agent. A more compact representation

can be built around test equivalence classes. A test is a possibly null string of actions followed by a color. Note that this is a simpler notion of test than the one used in section 4.2. To apply a test t to a state q means to execute the sequence of actions and to read the color of the resulting state qt . An empty test reads the color of the current state. [Rivest-87b],[Schapire-88] define *diversity*, a measure of reduction, based on equivalence of tests, that can be applied to such automata. Here is a simplified account. Two tests are equivalent if they give the same result in every state:

Definition 4.3.3 $(\forall t_1, t_2 \in \Sigma^*) t_1 \sim t_2$ iff $(\forall q \in Q) \phi(qt_1) = \phi(qt_2)$

Let $\langle t \rangle$ denote the equivalence class of t .

Let us look again at the universe of square tiles. Assume that c denotes the color of the tile faced by the agent. An example of equivalent tests is the pair **aalc** and **lalllac**. (see (2) above).

Definition 4.3.4 The diversity of the domain is the number of test equivalence classes

Definition 4.3.5 The assignment automaton (AA) is the tuple

$A' = [Q', \Sigma, K, \delta', \phi']$ where

$Q' = \{ \langle t \rangle \mid t \text{ is a test} \}$ is the set of test equivalence classes

$\phi'(\langle t \rangle) = k$ where k is the result of the tests in $\langle t \rangle$ when applied to the current domain state q

$\delta'(\langle t \rangle, b) = \langle bt \rangle$

Let us call the elements of Q situations, and those of Q' states. Note that AA has one state for each test equivalence class. Each state is labeled with the result of its equivalent tests. Therefore the set of all labels (the values of ϕ') represents the entire information that can be gathered by the through performing tests in the current situation.

The transitions of AA are arrows labeled b that go from the equivalence class of t to the one of bt . This enables AA to represent the initial automaton A . Here is why: If the agent performs the action b , going into a new situation, the test t in this new situation would yield the same result as the test bt performed in the old situation. Therefore, for every action b performed by the agent, we can easily recompute the new results of tests by assigning to $\phi'(\langle t \rangle)$ the old value of $\phi'(\langle bt \rangle)$. This is the basis of the simulation result. (see (4) below)

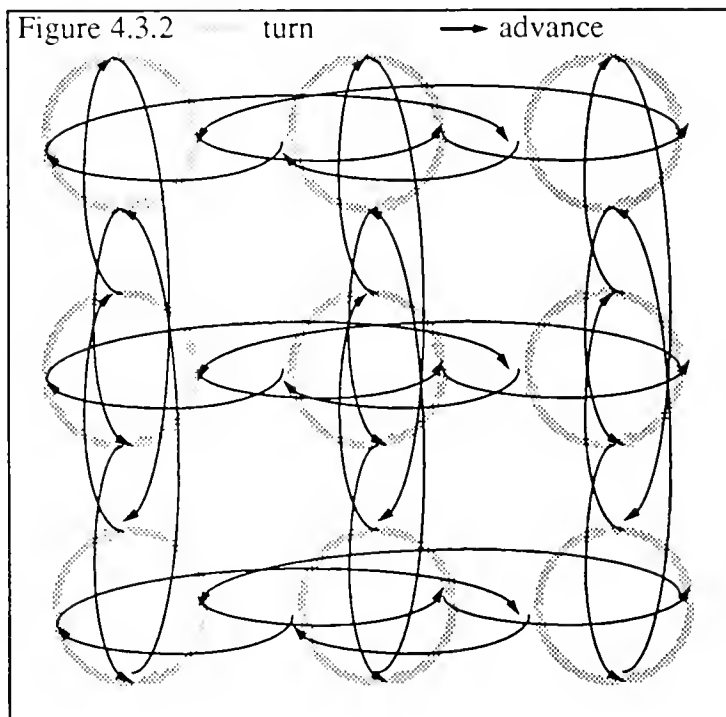
Definition 4.3.6 An automaton is reduced if for any two states there is a test that distinguishes them.

The environment diversity is the number of states in (AA). Let E be the original

environment, that is a reduced Moore automaton that represents it. [Rivest-87b],[Schapire-88] prove the following

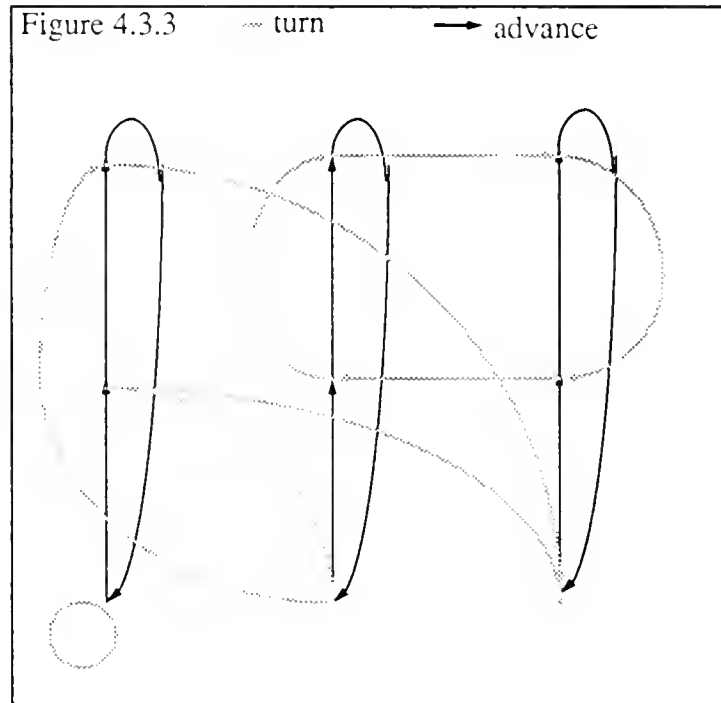
- (3) If n is the number of states of E , then $\log(n) \leq \text{diversity}(E) \leq 2^n$
- (4) If results to tests is what determines behavior, then AA can simulate the behavior of E . The current state of E corresponds to the class of the empty test in AA. The execution of an action 'a' affects the ϕ ' mapping: if $\delta'(q_1, a) = q_2$ then q_1 receives its new color from q_2 . Therefore some states in AA will get their colors reassigned. If action a is executed and there is an arrow labeled a that goes from q_1 to q_2 , then, the color of q_2 will 'travel' in the reverse direction along the arrow, and it will be assigned to q_1 .

Example 4.3.5 Resuming the universe of square tiles, assume that there are 3×3 tiles and this universe wraps around in both directions. A complete automaton with actions *turn* and *advance* is shown in figure 4.3.2. There are 36 states although there are only 9 tiles.



The corresponding assignment automaton is shown in figure 4.3.3. The reduction in representation size is dramatic. There are only 9 test equivalence classes, corresponding in fact to the 9 tiles. One should imagine the agent placed on the lower left state (the one corresponding to the null test). The agent is facing up. The black arrow

is pointing to the tile in front of it. The assignment automaton represents quite accurately the agent's perspective. An *advance* rolls the space like a cylinder, sliding the tile colors along the black arrows, from the point to the base. A *turn* is rotating the tiles such that the one that was at the right of the agent, appears now in front of it.



4.3.3 A Practical Exercise

Let us take a simple domain of the kind described in the previous sections and look in more detail at several approaches for an agent is learning it by exploration.

Example 4.3.5 The domain is a restriction of the one in example 4.3.1. There are two objects: a cubic block and a flat square. Opposing faces of the cube have the same color. The colors of the cube are A,B,C, the ones of the square are A,B. The actions available to the agent are h,v for horizontal and vertical rotation respectively and m for switching from one object to the other.

To unify our different attempts at exploring this domain, we would like to present the information to the learning algorithm as an infinite string. This string should contain both the actions and their results. A simple way to generate the string would be to choose actions randomly and alternate them with state 'colors'. Therefore each action will be followed by the color of the resulting state - the color 'perceived by the agent as a result of that action'. This also provides a general method to transform an exploration problem into a classical induction problem. The algorithm will have to separate the random component of this string from its structure.

Obviously, a minimal finite state automaton model exists for the domain. A complete one is shown in figure 4.3.4. It has 72 states and 84 transitions. Also, a finite memory model exists. One with a WORK tape size 3 is shown in figure 4.3.5. The transition schemata have been grouped in a graph. The representational advantage of FMM's is illustrated by fact that the three connected components of this graph correspond to the *conceptual*, mutually independent, regularities of the domain.

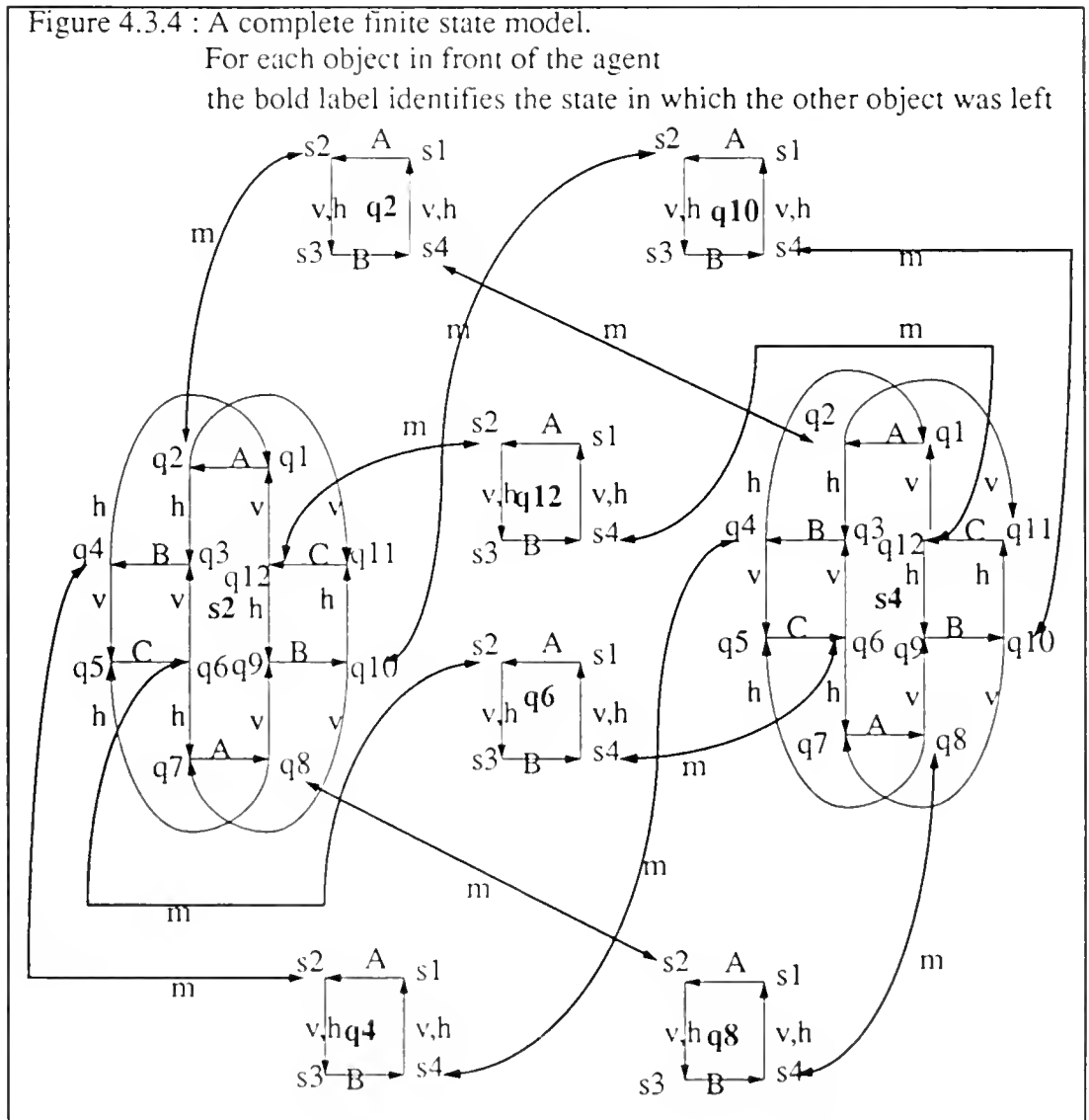
Following the library of models idea, if the universe to be explored has some deterministic subparts, we can start the learning process by accumulating only fragments of knowledge about those parts. The agent will have to recognize when one of these parts are reached. Once this is done, it will be able to predict the following steps until it leaves that known fragment. This is equivalent to wandering through a city of which we know some neighborhoods. The exploration will alternate three phases: *confusion*, *recognition of a familiar neighborhood*, and *prediction while moving on known territory*. That is the reason why libraries of routines will be more appropriate than single models. Each routine will correspond to a known area of the universe.

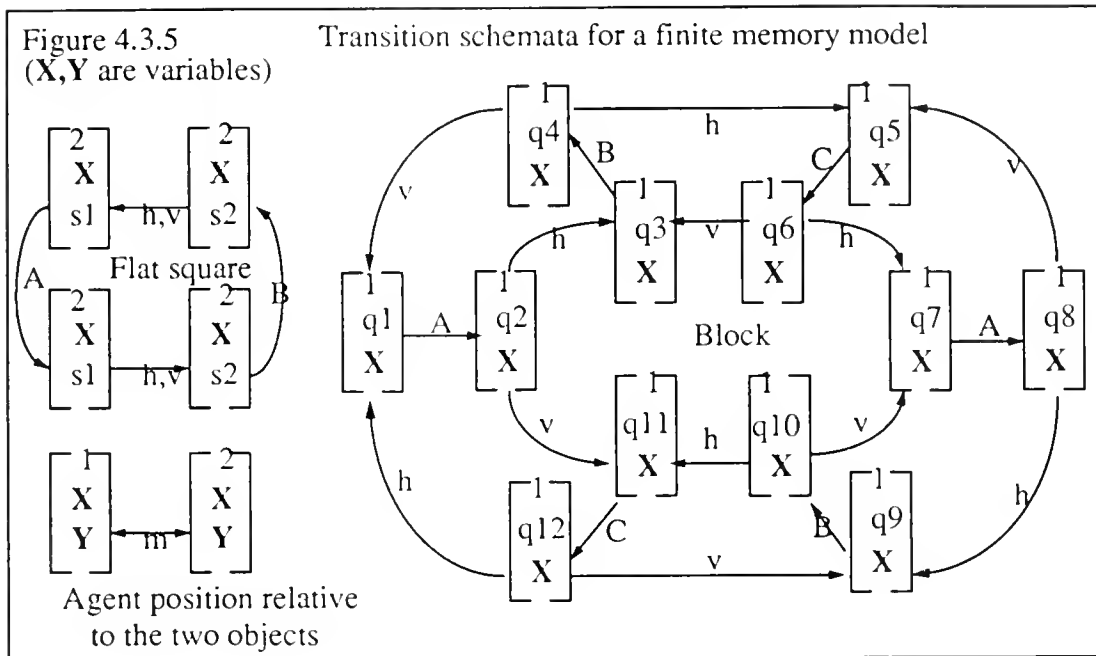
Since we shall use finite automata as routines, note that the concept of *id_sequence* presented in section 4.2 can be adapted to this approach too. There, an *id_sequence* identified a state of a complete model. We can relax this by having it identify a state of a partial model, that is a state in a subgraph equivalence class (see 4.3.2.a above). Let us call this one a *partial id_sequence*. Take for instance all the states marked s2 in figure 4.3.4. A sequence AhBhA encountered during the exploration will

restrict the possible states to exactly this set. The agent doesn't have to know that there are 6 different situations determined by this sequence. From its perspective, it is clear that the object in front of it is the flat square, and, as long as it doesn't move from it, the effects of **h** and **v** are perfectly predictable. The first **m**, however, brings again confusion until another partial id_sequence is encountered.

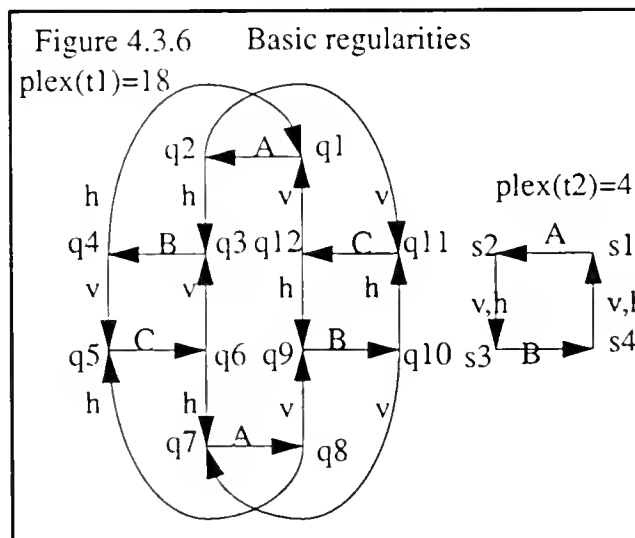
The smallest model set that achieves a significant compression is a singleton containing a two states six transitions machine that identifies the following regularity

$((A+B+C)(h+v+m))^*$ that is, the strict alternation of actions and colors.

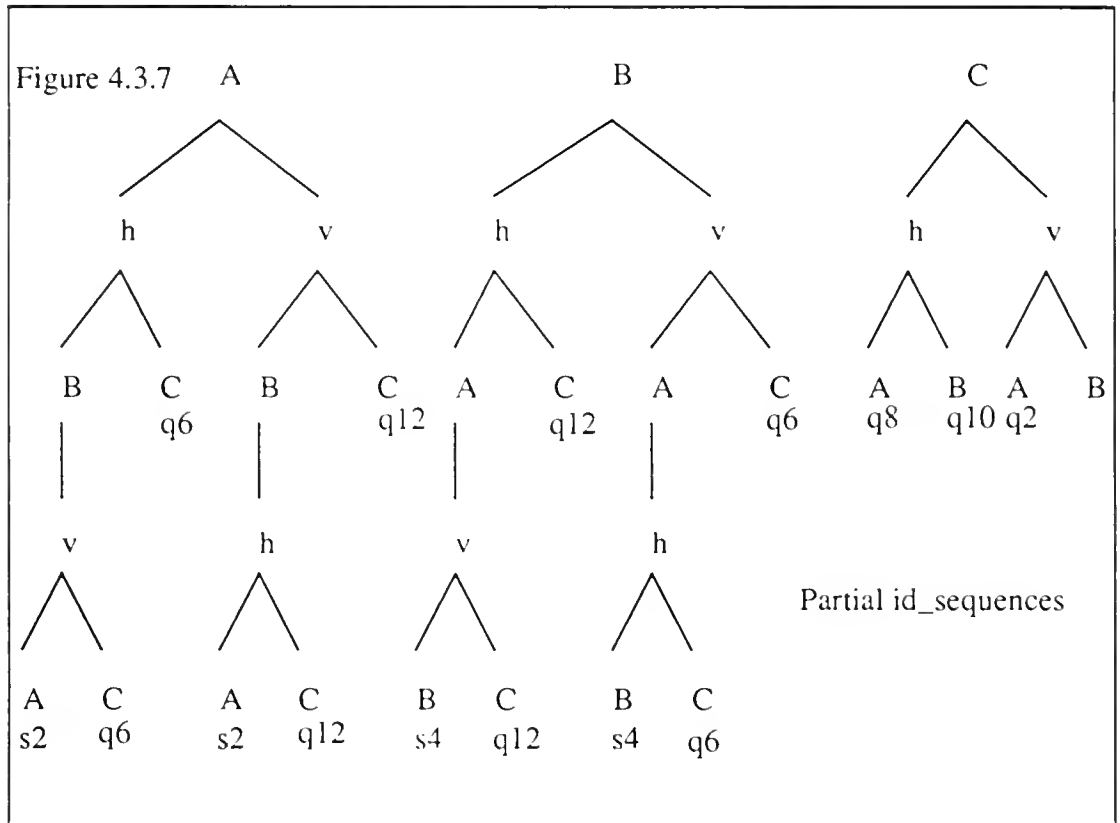




A first library of stable models to achieve a significant compression and parse a large percentage of the string length should contain the pair of automata shown in figure 4.3.6. In a long sequences of h and v actions, that is when the agent doesn't switch between the two objects, the two models in figure 4.3.6 are not competing for parsing the same substring. On the contrary, they alternate. On strings generated by interaction with the cube the first model will achieve the largest compression. On strings generated by the interactions with the flat square, the second model will perform better. They will both fail on m actions, that is in switching between objects.



A complete set of partial id_sequences is shown in figure 4.3.7. They identify the 2 possible states of the flat square and the 6 possible states of the cubic block.



A straight-forward enumeration of possible libraries is shown below (algorithm 4.3.1). In each hypothesis $h = [m_1, \dots, m_k, i, q]$ there is a currently running routine m_i and its current state q . Each routine $m_i = [Q_i, \phi_i]$ is a finite automaton with states Q_i and transition function ϕ_i . The initial automaton is one with a single state q_0 and no transitions defined. When one of the routine fails two groups of new hypotheses are added. In one group the routine is refined to fix the failure, in the second group the control is passed to another routine. The latter case also provides the occasion for adding a routine to the library.

(We are working on another algorithm that follows the ideas presented at the end of section 3.5. It is based on the assumption that the components of the library are substantially dissimilar)

Algorithm 4.3.1

```
H <- { h } where h = [m1, l, q0] and m1 is the initial automaton
Loop
  Let a be the next symbol to be parsed
  For all hypothesis h = [m1, ..., mk, i, q] in H
    If  $\phi_i(q, a) = q'$ 
      Then Replace h by [m1, ..., mk, i, q']
    Else Remove h from H
      (----- Create all refinements of mi that can parse a)
      For all j=1,...,|Qi|
        Create h' = [m1, ..., mi', ..., mk, i, qj]
          where mi' is mi with  $\phi_i(q, a) = q_j$ 
        H <- H  $\cup$  {h'}
      Let qnew be a new state symbol
      Create h' = [m1, ..., mi', ..., mk, i, qnew]
        where mi' is mi with  $\phi_i(q, a) = q_{new}$  and Qi' is Qi  $\cup$  {qnew}
      H <- H  $\cup$  {h'}
      (----- Jump to another running model, including a new one mk+1)
      For all l=1,...,k
        For all j=1,...,|Ql|
          H <- H  $\cup$  {h'} where h' = [m1, ..., mk, l, qj]
      Create h' = [m1, ..., mk, mk+1, k+1, q0]
        where mk+1 is the initial automaton
      H <- H  $\cup$  {h'}
```

This example shows that objects, if different enough are basic regularities of this class of domains and a proper representation of objects could emerge from induction performed on the agent's sensory-motor stream.

4.4 Model World 3 (Tiles-land)

4.4.1 The Class of Domains

We shall move now to a more complex class of toy worlds where the agent is endowed with a richer sensor. This world is a square covered by square tiles. It is an enriched variant of the toy world presented in [Drescher-86]. It is inhabited by an agent whose unique eye can encompass the entire square. Here are a few images that the agent can perceive:

Figure 4.4.1.1					
- - - - -	- h - - -	- - - - -	- - - - -	- - - - -	- - - - -
h - - -	- - - - -	- - - - -	- h x - -	- - h x -	- - - - -
x - x -	- x - x -	- h - x -	- - x - -	- - - x -	y h - - -
y - - - -	y - - - -	y x - - -	- - - - -	- - - - -	- - - - -
(a)	(b)	(c)	(d)	(e)	(f)

For our instance of Tile-land we shall consider that the eye perceives the entire world at one glance and that the tiles wrap around both vertically and horizontally. This is a flat world where objects appear as markings on tiles. The smallest object occupies an entire tile. The agent has a hand that is always visible and that occupies one tile.

The agent has no information about the spatial structure conveyed by these images. It perceives the square of tiles as a vector of unrelated sensory inputs without knowledge of horizontal or vertical adjacency.

[-, -, -, -, -, h, -, -, -, -, x, -, x, -, y, -, -, -, -, -, -].

The 'h' in the sensory image (sensory vector) stands for the hand. The agent can control it through the following repertory of actions

$A = \{ \text{up, down, left, right} \}.$

For example, in a situation where the perceived image is (a), applying 'up' results in a new perceived image: (b). We shall express this transition as

(a),up -> (b)

The world has certain regularities that the robot has to discover. They all depend on the obvious, but unknown to the agent, spatial structure. They can all be succinctly

expressed in English. The main question here is: How can the agent derive an appropriate representation for such regularities if it starts with something as simple as action, sensory vectors and transitions?

This is a small sample of regularities that can be postulated about Tile-land.

- (i) The hand's destination depends on its starting position and the action. The content of the destination tile may prevent the move. Otherwise, the hand's move does not depend on the content of other tiles.
- (ii) Actions are, with some exceptions, commutative :
(s),up,left -> (s') \Rightarrow (s),left,up -> (s')
- (iii) An isolated *x* is an object that can be *pushed* by the hand :
(a),down -> (c)
- (iv) A group of adjacent *x*'s is a unique, rigid object. It can be pushed by the hand like a one tile object: (d),right -> (e).
- (v) An object marked with *y* is a fixed obstacle that cannot be moved :
(f),left -> (f).

Simple but interesting variations on Tiles-land can be obtained by adding the following features:

- (vi) Allow the hand to move in two planes and provide actions to switch between them:

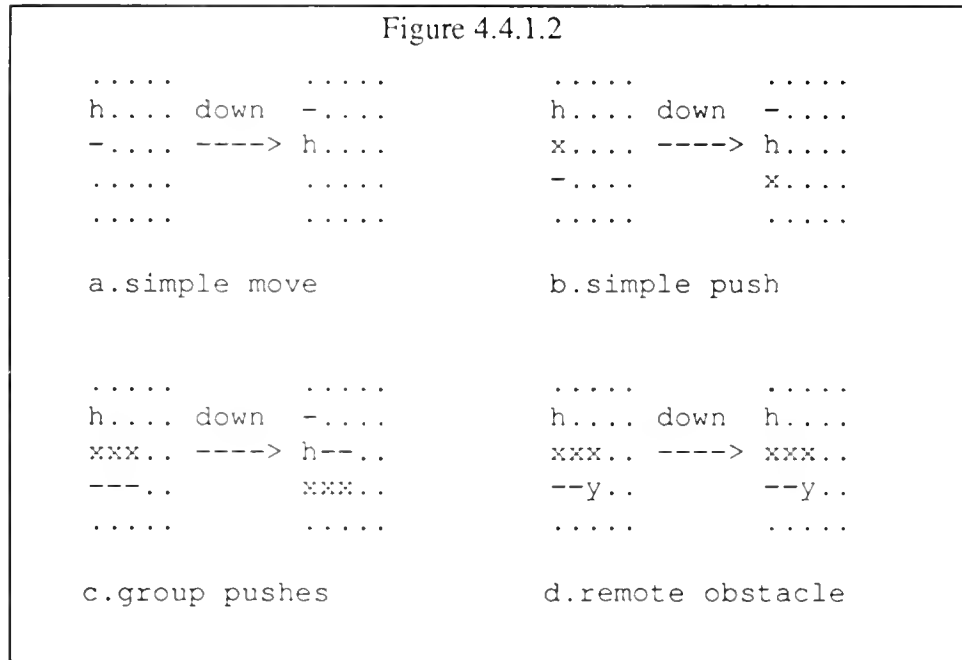
{ lift-hand, lower-hand }

The hand will be able to move over other objects. As a result, some tile markings may be temporarily hidden by it.

- (vii) Allow the hand to { *grab*, *release* }. These actions will affect the object under the hand. By being grabbed and lifted and object is temporarily brought to the upper plane and can be moved over other objects.

- (viii) Provide a larger world and let the eye perceive only a small square region of it. Provide actions for eye movement too. One of the consequences of this is the fact that the hand will not be always visible.

Some instances of the above rules are represented below, using the dot '.' as a "don't care" symbol:



Note that in figure 4.4.1.2.c, the hand movement causes the downward translation of an object seen as a horizontal group of three adjacent x markers.

Starting only with tiles, markers, and adjacency relations, the presence and position of such objects can be expressed by a simple computation. Therefore, note that the transitions in figure 4.4.1.2 can be expressed, given an appropriate language, as

condition_1(state) -> condition_2(result(state,move))

a desirable, general form, usable for planning or navigation purposes, as mentioned in the introduction to this chapter. For such rules to accurately express transitions we only have to add a tacit assumption: The content of tiles not affected by the applicable rules remain unchanged.

Note, also, that a *context dependent computation* (with working tape size limited to the size of the sensory input) is enough to express all the above rules most succinctly. With a couple of supplementary symbols used as markers (in the spirit of Ullman's visual routines [Ullman-85]) we could specify a simple program that, given the hand move, would work directly on the starting image and transform it into the resulting image.

Finite memory models can represent this domain too, for the same reasons. Consequently finite automata are equally appropriate. Still, they would not be able to cap-

ture in a simple routine the concept *connected, rigid, group of x 's* that is necessary for the perception of objects.

Let us now focus on the construction of appropriate representations. We have to start from an initial representation of images, actions and transitions. Therefore we have to introduce constants to name the initially unrelated sensory elements. Since we assume that an image encompasses the entire surface, and the eye does not move, the same constants will also provide unique names for tiles:

Figure 4.4.1.3

a1	a2	a3	a4	a5
a6	a7	a8	a9	a10
a11	a12	a13	a14	a15
a16	a17	a18	a19	a20
a21	a22	a23	a24	a25

A pair like [a2, h] will represent the fact that the sensor a2 gives the indication h, or that the tile a2 contains the marker h. An entire image will be a set of 25 such pairs. A smaller set of pairs naturally represents a predicate that may be true of several images.

4.4.2 Building New Representations Based on Co-occurrence of Attributes

The first representation modification will consist in learning the tiles adjacency relations that govern all moves. [Drescher-86] approaches the same problem in the framework of a computational equivalent to Piaget's theory of sensory-motor development in infants. Drescher allows the agent to move both hand and eye and attempts to detect a statistical correlation between tile content before and after an action. He proves that such correlations can be learned from arbitrary transitions, and that they constitute a basis for the agent to learn a hand-eye coordination. Obviously, tile adjacency is the underlying concept that is learned in the process. We shall approach this problem from a more general perspective.

Our first goal will be to uncover statistically stable relations among parts of descriptions. Then we investigate how such relations can be used to change and compress the descriptions, and by this, construct new representations for the sensory-motor stream.

As long as descriptions were strings, the concatenation imposed an order of presentation for the string's symbols. This, in turn, determined $a_1 a_2 \dots a_k$ to be the basis for predicting the symbol a_{k+1} . Since we assume here a representation of the sensory image as an unordered set of elements, we shall attempt to detect any dependencies among set elements.

(a) Finite Presentation of Sets

We are shown the subsets S_1, \dots, S_n of a given finite universe U . Say S_i are described by binary vectors of length $|U|$. We shall call the subsets S_i presentations. The content of these subsets is determined by unknown regularities. For example, some elements of U may be consistently correlated, that is, the presence some of them in any presentation S_i may determine the presence of others in the same presentation.

The induction problem, that refers now to a string of sets, is to predict the presence or the absence of U 's elements in a future S_{n+1} , based on the presentation S_1, \dots, S_n . In other words, to discover the unknown correlations.

One type of regularity to consider first concerns the probability distribution of the elements of U . For an element $x \in U$, $P(x)$ will be estimated by the frequency of the event $x \in S_i$ for $i=1, \dots, n$. A nonuniform distribution of $P(x)$ is a regularity of this type.

A second type concerns dependencies among elements. Let the event A stands for the presence of the subset A , that is $A \subseteq S_i$ for $i=1, \dots, n$. Any difference between

$P(x)$ and $P(x|A)$ is such a regularity. The events $\sim x \wedge \sim A$, $\sim x \wedge A$, $x \wedge \sim A$ should be considered too. However, for the purpose of discovering some of the rules of Tileland, it will be sufficient to look for a stronger type of dependency, characterized by $P(x|A)=1$.

Since we are shown only a finite number of subsets, a systematic study may enumerate such dependencies in increasing order of model complexity. Here, we could estimate that complexity as the logarithm of the number of elements involved.

One possible approach would go like this:

Define a binary relation R_2 in $U \times U$ $R_2(x,y) \iff (\forall i)(x \in S_i \Rightarrow y \in S_i)$

$R_2(x,y)$ means that *in every presentation the presence of x determines the presence of y* . Note that R_2 is reflexive and transitive by definition. One thing to do is to study the properties of R_2 , i.e. check the following:

- (i) Are any distinct elements related? $(\exists x,y \in U) x \neq y \ \& \ R_2(x,y)$
- (ii) Is every element related to another? $(\forall x,y \in U) R_2(x,y) \vee R_2(y,x)$
- (iii) Is R symmetric? $(\forall x,y \in U) R_2(x,y) \Rightarrow R_2(y,x)$
- (iv) Is R antisymmetric? $(\forall x,y \in Y) R_2(x,y) \Rightarrow \sim R_2(y,x)$

If R_2 is a partial order, then represent S_i by $\min(S_i)$

If R_2 is an equivalence then represent S_i by S_i/R_2

If R_2 is empty (except for the pairs $\langle x,x \rangle$)

then look for dependencies among triples:

Define a 3-ary relation R_3 in $U \times U \times U$ $R_3(x,y,z) \iff (\forall i)(x,y \in S_i \Rightarrow z \in S_i)$

etc.

In general, the problem amounts to studying the binary relation r on $2^U \times U$ defined below:

$R(A,x) \iff (\forall i)(A \subseteq S_i \Rightarrow x \in S_i)$

$r(A,x) \iff R(A,x) \ \& \ (\forall B \subset A)(\sim R(B,x))$

This minimization is needed because, by definition, if $A \subset C$, then $R(A,x) \Rightarrow R(C,x)$, but $R(C,x)$ is redundant, therefore we want $r(A,x)$ to hold only for the smallest sets A that determine x .

For our case, however, we cannot limit the study to relations that have such

strong properties like (ii),(iii),(iv). Besides, we cannot assume a finite presentation either, since our agent continues to move and to receive new sensory images, even in a finite world. The only assumption that we can rely on is the fact that U , the set of all possible sensory events, is finite.

(b) Infinite Presentation of Sets.

In a sensory-motor stream the sets S_i are presented as an infinite sequence. At least, the agent has to start guessing before seeing them all, thus an incremental algorithm is needed.

A related situation is presented in [Paturi-89]. Here is the problem: There are n electrical bulbs that can switch between *on* and *off* independently every Δt seconds. It is known that there is a particular k -tuple of bulbs that are statistically correlated, that is the probability of them being in the same state together is substantially larger than that computed on the assumption of independency. [Paturi-89] approaches this in the framework of PAC learning defined in [Valiant-84]. He gives several polynomial time algorithms that, under certain conditions, given an error margin ϵ , will find, with probability $1-\epsilon$, the most highly correlated tuple.

Here though, we are concerned with a deterministic correlation, that is a set of values present in an example always determines the presence of another value. We only need the existence of an algorithm to detect such dependencies in order to prove the validity of our induction scenario. A simple one, for example, would keep a set of counters c_A , one for every subset A of events. In other words, the variable c_A counts the number of times the subset A was part of a presentation S_i . This allows the approximation:

$$P(x|A) = c_{\{x\}}/c_{A \cup \{x\}}$$

With the assumption of uniform distribution for each random variable x , we can compute a level of confidence for a given dependency after n observations. Using the concepts introduced in chapter 3, a model (i.e. an induction hypothesis) can be represented as:

$$[A \rightarrow x].$$

When an invalidating observation is encountered, that is a presentation S that includes A but not x , the model can be generalized to

$$[A \cup \{y\} \rightarrow x] \text{ for all } y \in U - (A \cup \{x\})$$

This allows for an induction by refinement algorithm that considers statistical dependencies in the increasing order of complexity. For example, note that a simple push may be first perceived as a dependency among action, hand starting position, and ending position. An instance with an obstacle object in the ending position will invalidate it. In this case, the refinement shown above, will generate, among others, a dependency with one more necessary condition: a blank on the destination tile.

Later in this section we shall construct more complex concepts based on dependencies. We shall live aside concerns about how to retract those concepts when a dependency becomes invalid. We shall only assume that the construction of each level of concepts will start only after the confidence on the lower one is high enough.

(c) Sets of Tuples.

So far the elements of U were unanalyzable, atomic sensory events. However we would like to take advantage of the entire information provided by the sensorial representation in Tiles-land. There we can distinguish between sensors and values of sensors. Also, we would like our scenario to start at the level of transitions, that is, triples [sensory-image, action, sensory image]. Therefore we have to consider the following, more general formulation.

Let U be a universe of tuples over $A_1 \times A_2 \times \dots \times A_k$, hence the examples S_i are sets of tuples. Here we can look for the same correlations as before. For example, if the tuples are pairs, $r(\{[a,b],[c,d]\},[a,e])$ means that whenever $[a,b]$ and $[c,d]$ are both present in an example, $[a,e]$ is present too. We can write this more suggestively as

$$\{ [a,b],[c,d] \} \rightarrow \{ [a,e] \}$$

Besides, through λ -abstraction, we can also consider more general correlations schemata like

$$\{ [a,X],[b,X] \} \rightarrow \{ [c,X] \}$$

where X is a variable that stands for all values of the second component.

This last formulation of the problem can be applied to a learning agent that is endowed with a vector of sensory inputs, each with a preestablished set of values. This agent views the world as a sequence of sets of pairs $[s,v]$ where s is a particular sensor and v is its value for that time step.

If the agent is not passive, i.e. is endowed with a set of actions, it can also view the world as a sequence of transitions *situation,action->situation*. These transitions can also be represented as sets of triples:

[x,s,v] where

x ∈ {before,after} identifies the situation,

s ∈ {0,1,2,...,m} identifies the sensor elements (1,2,...,m) or
the action element (0),

v stands for the value of the element s.

Note that the tiles adjacency that governs the simple moves is precisely such a stable dependency among the values of three positions in a transition vector: initial and final positions of h and the move. The same is true about simple pushes.

Let us assume that all instances of the stable relation implied by the simple pushes have been presented many times and the corresponding dependencies have been stored in a memory. For example, the vector:

[.h...x...] down [.....h...x...]

is stored as the dependency:

{[before,a₂,h], [before,a₇,x], [before, action, down]} => {[after, a₇, h], [after, a₁₂, x]}

All the discovered dependencies can obviously be used together as a *library* model to obtain compressed descriptions of transitions. Nevertheless, we shall give up the idea of a complete representation for past transitions.

Let us assume that only individual dependencies are stored in the agent's memory as they are discovered. After presenting enough simple push transitions, the memory should contain a complete description of the spacial structure of Tileland, that is, the *horizontal and vertical adjacency* relations between tiles.

(d) Schemata

We shall look now at another level of induction that finds models for sets of schemata. The models considered here are λ-abstraction of correlations or correlation schemata. For example:

{ [before,a₁,h], [before,action,right] } -> { [after,a₂,h] }
{ [before,a₇,h], [before,action,right] } -> { [after,a₈,h] }
{ [before,a₄,h], [before,action,right] } -> { [after,a₅,h] }

are covered by the model r₁

$$r_1(X,Y) = \{ [before,X,h], [before,action,right] \} \rightarrow \{ [after,Y,h] \}$$

The same λ -abstraction works recursively on schemata. For example:

$$\begin{aligned} \{ [before,X,h], [before,action,right] \} &\rightarrow \{ [after,Y,h] \} \\ \{ [before,U,h], [before,action,left] \} &\rightarrow \{ [after,Z,h] \} \\ \{ [before,V,h], [before,action,down] \} &\rightarrow \{ [after,W,h] \} \end{aligned}$$

are covered by

$$r_5(A,B,C) = \{ [before,A,h], [before,action,B] \} \rightarrow \{ [after,C,h] \}$$

Therefore the agent's memory could maintain a hierarchy of schemata, like:

$$r_5(A,B,right) \Leftrightarrow r_1(A,B)$$

$$r_5(A,B,left) \Leftrightarrow r_2(A,B)$$

Note that the maximum compressibility models will avoid over generalization through abstraction since a model that is too general will need more determinants (i.e. variables) to specify its instances, so the corresponding codes will be longer.

4.4.3 Induction Scenario

So far we considered, here and in the previous chapters, many variations on the idea of refining a hypothesis in order to adapt it to new information. For example, we considered the *library* models as classifications and we proposed an algorithm (3.4.2) to decide to which class (i.e. routine) should we assign a new data item.

Here too, we consider that we have accumulated in a memory all simple dependencies, that is, the entire tile adjacency relation. We can see them as transition prototypes, since they do cover all simple pushes. During exploration, more complex prototypes may be accumulated in the memory. When a new item was presented we used to ask the question: *should this become a new prototype or is it covered by an old one?*

In Tiles-land, we shall consider a new approach. This is to attempt to express the new item, say a new transition, as *a computation that includes several old prototypes*. Once we found it, that computation will become a new prototype.

To see how this might work, let us take an example. Assume that we present to the agent a group push where the x's are aligned on the direction of the push. Say, for

instance, the vector

$$(1) [-h---x---x---x-----]down[-----h---x---x---x---].$$

The same mechanism of uncovering correlations can eventually find a rule on pushing down groups of three x's. However more complex computations could emerge if we allow a new relation to be expressed as a composition other known relations. The problems with this are:

- (a) How to justify a more complex expression instead of a simple one?
- (b) How to find whether such a composition exists?

To answer (a), we shall look at an abstract situation. Let us consider that we find, for a certain instance, i.e. for a certain choice of constants c and d, that

$$(2) f(c,d) = g(c, g(h(c), g(h(h(c)), d))).$$

Some compression may be achieved by a possible generalization of constants. in this case, having already stored g(x,y) and h(z) for all the values of x,y and z, we would only have to store the formula for f:

$$(3) f(X,Y)=g(X,g(h(X),g(h(h(X)),Y))).$$

Note that one can make an analogy between the construction of a formula for f based on the predefined g,h and the construction of an explanation in Explanation Based Learning (EBL). Turning the constants into variables allows a type of generalization similar to the one used there. Like in E.B.L (discussed in the next chapter) more regularity can sometimes be extracted from the explanation. The formula (2), for example, also suggests a few generalizations based on recursion:

$$(4) f(X,Y) = \text{if } p(X) \text{ then } g(X,Y) \text{ else } g(X,f(h(X),Y))$$

$$(5) f(c,d) = u(c,d,2) \text{ where} \\ u(X,Y,N) = \text{if zero}(N) \text{ then } g(X,Y) \text{ else } g(X,u(h(X),Y,N-1))$$

Therefore, once certain simple relations have been established and memorized, (adjacency in our case) a subset of a primitive recursive functions family could be gradually constructed in order to account for more complex situations. Their discovery corresponds to the emergence of more complex concepts that are added to the descriptive language. The only criteria, for preferring a complex formula for a new item instead of directly memorizing it, is the contribution to the total compressibility achieved by the agent's memory.

In other words, instead of (2) we can use (5), that is $u(c,d,2)$. We are justified to do so if $u(X,Y,N)$ covers enough instances to achieve compression. In this case $u(X,Y,N)$ becomes a new prototype, a new concept.

In order to answer (b) we have to give details on:

- (b1) How can a composition formula be found?
- (b2) How can it be generalized?

(b1) We start from the fact that both the newly presented item and the relations already stored in memory can be viewed as sets of triples. We shall look for a path that connects some elements I of the new item to other elements O of the same item such that I uniquely determines O . We shall require this path to go through relations stored in memory. That is we shall look for a candidate formula for a function $O=f(I)$, that computes some elements of the item from the others, via already known relations.

Going back to our specific item (1) let us express it as a dependency:

(6)	[before, a_2 , h]		[after, a_7 , h]
	[before, a_7 , x]		[after, a_{12} , x]
	[before, a_{12} , x]	=>	[after, a_{17} , x]
	[before, a_{17} , x]		[after, a_{22} , x]
	[before, action, down]		

Our answer for a (b1) would be to find a way to compute the right-hand side of (6) from the left-hand side, using for this purpose already known dependencies. Finding the proper stored dependencies is, in this case a problem of matching. We show this in figure 4.4.1. We can visualize the construction of these matches as a recursive associative recall from memory.

Imagine a process (spreading activation, marker passing, etc.) that starts from all the triples of (6), and primes an associative memory in order to find *similar* triples in memory. The recall is recursive, since the retrieved relations are further used as primers. (For one of the adequate associative architectures, see [Kanerva-87])

A first instance of a simple push is retrieved based on the following common elements:

[before, a_2 , h], [before, a_7 , x], [before, action, down], [after, a_7 , h], [after, a_{12} , x]

A second simple push instance could be retrieved from the first simple push and two elements of (6)

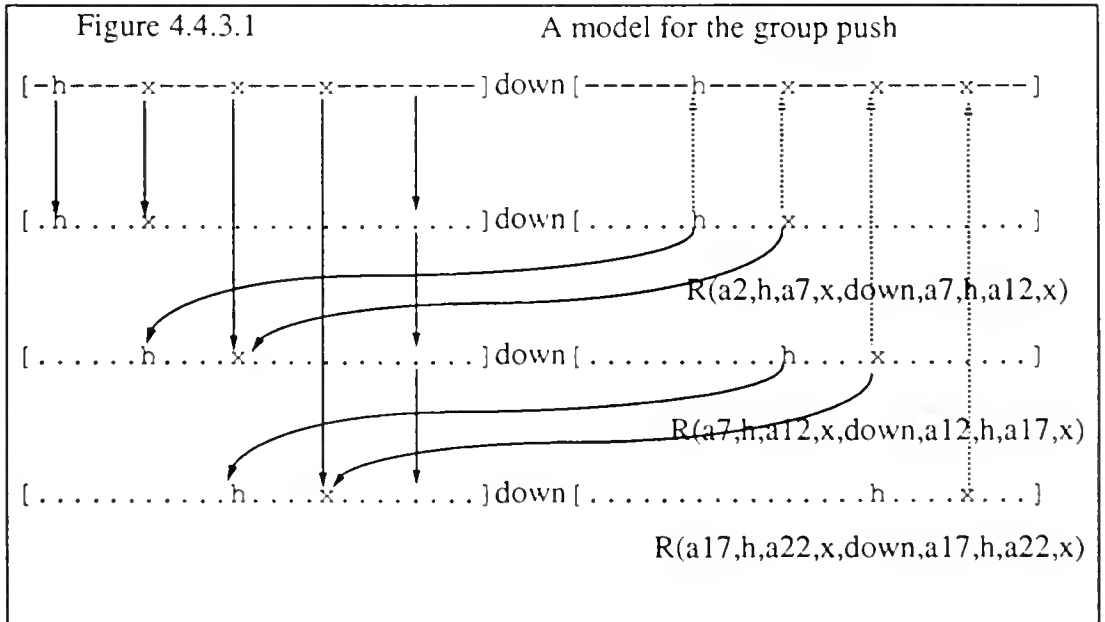
[before, a_{12} , x], [after, a_{17} , x]

This second retrieval involves matching similar elements:

[before, action, down] <--> [before, action, down],
 [after, a₇, h] <--> [before, a₇, h],
 [after, a₁₂, x] <--> [before, a₁₂, x]].

A third simple push is found from the second one. The process may stop here because all the elements of (6) are matched. The associations used in retrieval are shown in figure 4.4.3.1. The inputs of the new function are solid arrows, the outputs are dashed arrows.

This results in building a graph that ties all these associated relations together. The edges of this graph represent dependencies among elements. In our case they constitute an actual deterministic computation that consume certain elements of the new item and outputs other elements of it.

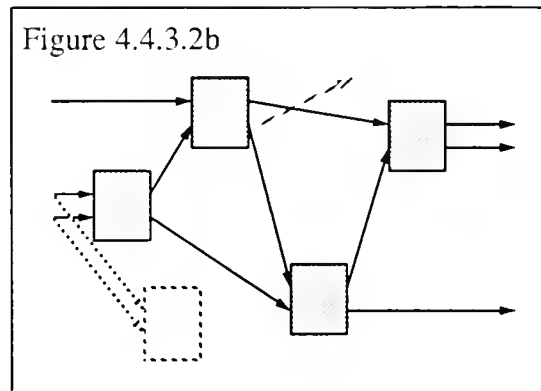
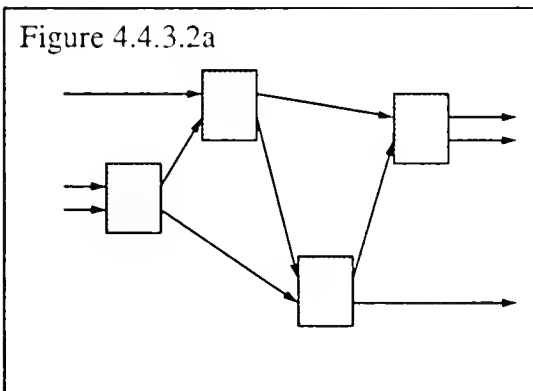


If the constructed computation is fully deterministic (as in figure 4.4.3.2a), that is every associative recall has a unique result, then the computation is a newly discovered stable relationship that uses the input elements to predict the output elements.

If there are more than one outcome in a retrieval (or the retrieved relations - represented by the gray squares - are not functions themselves) (dashed lines in figure 4.4.3.2b) then we can still use the entire structure as a predictive computation but we need to specify the choices made at the nondeterministic points!

Therefore, in the nondeterministic case we need another set of inputs: *the choices*. If we regard the entire computation as a *model* for a new item, then the sequence of choices is a *code*.

To summarize, nondeterminism in this scenario can occur either because of the associative retrieval (ex. several stored memory sets match the priming one), or because the matching elements of the retrieved relation do not uniquely determine the remaining elements (ex. the relation $\{u, v, w\} \Rightarrow \{x, y\}$ was retrieved based on matches on u and x ; v, w and y may not be determined by u and x , although x and y are determined by u, v and w).



As more relations are retrieved and added to the association graph, the instances of nondeterminism may accumulate. Each one of them implies a lengthening of a necessary disambiguation code. After a while, the combined complexity of this code and of the growing graph may exceed a threshold over the complexity of the new item itself, beyond which the association is arbitrary. Then, the recollection process may be deemed wasteful and could be abandoned.

At this point, we would like to mention another equivalence to explanation-based learning (EBL). The constructed computation is a way to derive the new item from the memorized ones. This corresponds to the explanation in EBL. There, what is constructed is a logical proof. The basic structure that supports this analogy will be presented in section 5.1 and further exemplified in chapter 5. The generalization issue, discussed next, has its EBL analogue too. This will become obvious in section 5.2.

(b2) How to generalize the newly constructed computation? For this, note that the computation (provided that it is a computation) could be further examined for compression, following the strategy of stepwise induction.

First, a λ -abstraction is possible through changing constants into variables. For

our example, this is further facilitated by the fact that all the nodes of our computation graph are instances of the *simple push*. Figure 4.4.3.3.a shows the graph with the appropriate variables denoted by capital letters. The letter R denotes the simple push.

Any computation is built from a graph of associated memory elements. This graph can be seen as a trace of that computation. It may be an instance of a recursive structure. A second learning level (a separate process that works on memory elements only) will identify it with the simpler graph in figure 4.4.3.3.b

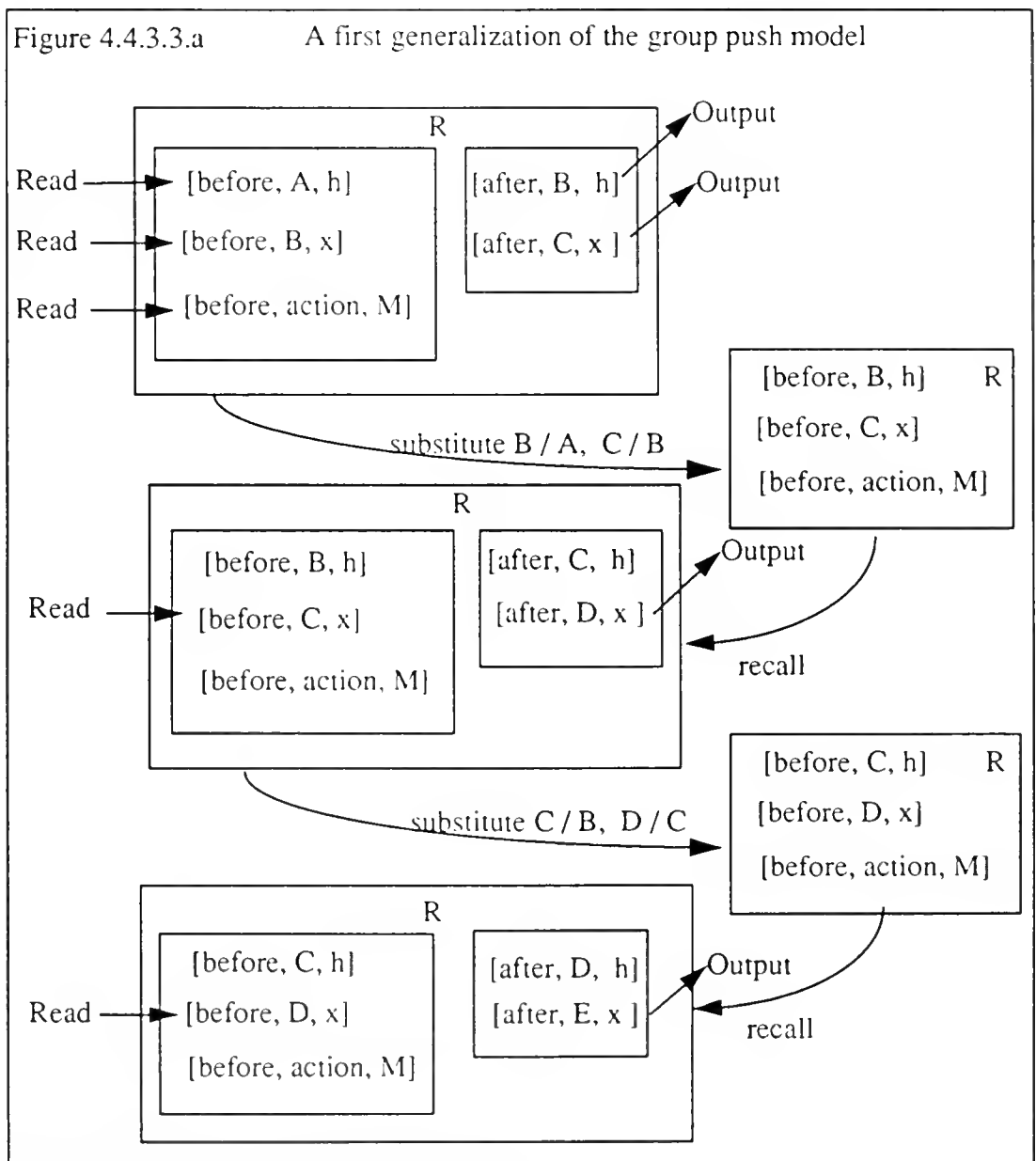
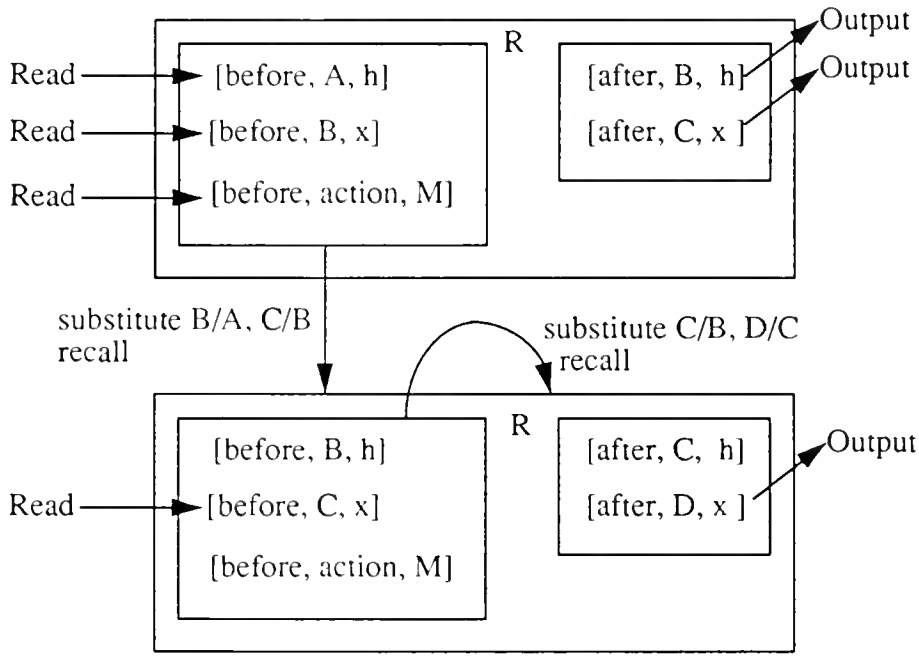


Figure 4.4.3.3.b A second generalization of the group push model



We can now attempt to define a compression factor for a stable relation or for a computation based on such relations. We shall see the relation or the computation in terms of a set of input variables X and a set of output variables Y . Let C be the set of choices that have to be specified in order to make the computation deterministic. In this case the relation/computation becomes a predictive rule R

$\text{condition}(X, C) \Rightarrow \text{predict}(Y = \text{function}(X, C))$.

The compression factor is defined by $\text{press}(R) = (|X| + |C| + |R|) / (|X| + |Y|)$

The above scenario supports several points:

- (i) Simple learning mechanisms (discovery of relations based on co-occurrence of elements in sensory vectors or transitions, associative retrieval of previous relations) could generate a virtually infinite family of computations of increasing complexity
- (ii) Creating another learning process, as soon as there is enough output to study from a previous one, allows more complex regularities to be discovered, as discussed in section 3.6.

(iii) Sensory representations of the transformations of the outside world may become internal templates of computations and may be generalized by application to new situations and by relating them to one another.

5. Background Knowledge and the Uses of Memory

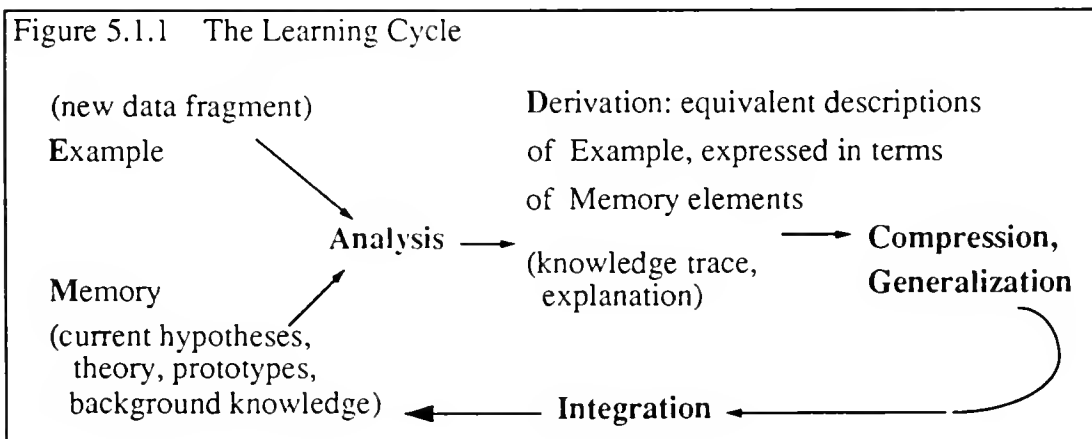
In this chapter we return to the question: how are models refined in order to incorporate new data? We define analysis as an essential component of learning and we present several instances of it. We relate it to the concepts of generalization and specialization. We discuss then two extreme cases of analysis. One is highly formalized and occurs in Explanation-Based Generalization. The concept of encoding through a model casts new light on the method. It allows us to establish its generality and to understand it as a form of focussing (specialization) of too general hypotheses. The second has many ad hoc implementations and occurs in constructive induction. Here we define the process of constructive change of representation and we outline its problems. We illustrate the issues in the context of Bongard problems.

5.1 The Learning Cycle

All our investigations so far entailed keeping a current set of hypotheses at every moment during the learning process. In general, at any moment, a learning algorithm should represent somehow what has been learned so far. Compression of future data will have to attempt to relate that data to the data seen before or to an encoding of it. Therefore it needs a *Memory*. The *Memory* contains at least the minimal a priori bias: the learning algorithm itself, and the particular class of models that it is able to produce. During learning the Memory contains the background knowledge against which the new data is analyzed.

Another a priori bias is representation used for the input data. Assume for instance that the data presented to it consists of descriptions of graphs. Then, the syntax and semantics of the particular representation used for those graphs is background knowledge that can be integrated into the algorithm. Consequently, it will work at a different conceptual level, dealing with nodes, arcs and labels, rather than with a simple string of symbols.

Figure 5.1.1 The Learning Cycle



Let us look at figure 5.1.1 as a summary of the induction scenario from the previous chapter. This is how transitions are learned in Tiles-land. When a new transition is presented, it is first analyzed in terms of existing transitions or transition schemata. One of the new expressions obtained as a result of analysis (the graph in figure 4.4.1) is chosen and generalized into a new schema or prototype. This is then stored in memory, increasing the collection of prototypes.

This cycle seems to us to underlay the structure of a large number of learning algorithms. It can be understood in terms of the algorithmic information theory. The cycle starts with some new data fragment that we shall call *example* (E). The first thing done to it is to relate it to the *Memory* (M). *Analysis* is the process of establishing the relationship between E and M.

The result of *Analysis* is a *derivation* (D) of the new fragment in terms of M, that is, a program that computes E from something known before. By relating E to M, Analysis has to determine what part of the example is the already known and what part is really new. These components are made explicit in D.

Proposition 5.1.1 In the perspective of the algorithmic information theory, the purpose of *Analysis* is to separate $K(\text{Example})$ into

- (i) $K(\text{Example} : \text{Memory})$ the mutual entropy, the part that is known,
- (ii) $K(\text{Example} | \text{Memory})$ the conditional entropy, the part that is new.

Using the concepts of chapter 3, if we take memory to be a hypothetical model, then analysis is the parsing of the example. Parsing may or may not succeed. If it does, we have a code, that is a complete derivation of the example. If it fails we have some partial derivations, a set of failure points and a set of unparsed fragments.

Since the code is the part that needs to be added to the model in order to produce the example, it can be assimilated to (ii), that is, the novelty. If the parts of the model that participated in parsing can be identified (for example certain productions in a grammar) they can be assimilated to (i), the shared information. These outcomes can be used to modify the model. If the model is not supposed to cover the example, then the information in (i) must be removed from it. If the model does not parse (cannot generate) the example, but should, the information in (ii) must be added to it.

These modifications are directly related to the traditional hypothesis refinement approach, that is based on the following concepts: H and E are sets of hypotheses, and examples respectively. They are related through the predicate covers. For every $h \in H$ and $e \in E$, $\text{covers}(h, e)$ stands for *h is a valid model for e*. The predicate induces on H a partial order \leq

$$h_1 \leq h_2 \Leftrightarrow \{e \mid \text{covers}(h_1, e)\} \subseteq \{e \mid \text{covers}(h_2, e)\}$$

where $h_1 \leq h_2$ stands for h_1 is *more specific* than h_2 , or h_2 is *more general* than h_1 . We also assume two functions: $\text{generalize}, \text{specialize} : H \rightarrow 2^H$ with these properties:

$$\begin{aligned} h' \in \text{generalize}(h) &\Rightarrow h \leq h' \\ h' \in \text{specialize}(h) &\Rightarrow h' \leq h \end{aligned}$$

To learn a concept from examples means to find a subset of hypotheses that cover all the positive examples presented so far, but no negative example. Therefore, algorithms based on refinement have to

down-refine, or *specialize*, the hypotheses that cover some negative examples, *up-refine*, or *generalize*, the hypotheses that do not cover all positive examples.

A refinement approach depends on the following issues: Are the sets $\text{generalize}(h)$ and $\text{specialize}(h)$ finite?, enumerable?, computable? Will the algorithm converge to a finite subset of H ? Section 3.3 gives some details. Related discussions are found in [Mitchell-78],[Laird-87],[Shapiro-81],[Utgoff-84]. A special problem appears when no negative examples are provided. How to avoid over-generalization? [Angluin-80], [Berwick-86] show that, to insure convergence, a minimal generalization has to be defined. Another approach to positive only examples is offered by a specific case of analysis, explanation-based generalization, that will be discussed in the next section. This table summarizes the connections between refinement and analysis :

	Derivation succeeds result: a complete derivation	Derivation fail result: partial derivations failure points unparsed fragments
positive example	Possible specialization (sect. 5.2) Generalize the information in $K(e \mid m)$, by selecting a part of it and add it to the model	Generalization Create submodels for the unparsed fragments, or extend the current model to parse them
negative example	Specialization remove from the model the part containing the shared in $K(e : m)$	Leave the model unmodified

Analysis can be identified in a variety of processes:

(a) Simple accumulation of information that has to be represented in some canonical reduced way. Take a system of equational constraints, say linear inequalities. We are given a new equation. The questions: does it add something new? does it contradict previous constraints? are legitimate. If it is derivable from the others, it is not worth keeping. Even if it is not, it may still be reducible to a more succinct formulation. Analysis is the process of answering these questions.

(b) Symbolic integration as performed by LEX [Mitchell-83]. We have a general search procedure and we are given a new problem. Applying this procedure is performing analysis. Failure means that the problem is not in the class covered by the general search algorithms. The possible subproblems solved identify known parts of the new problem. The search paths that failed and the unsolved subproblems contain the novelty that has to be added to our experience in order to increase the class of solvable problems. Success results in a tree of solution steps. The operators used were known but the structure of the solution tree is the novelty. It can now be generalized into a solution schema.

(c) Consider now a library of such general solution schemata and a new problem. Analysis may have to combine several schemata into a unique computation. The structure of this computation together with the instantiations operated on schemata contain the novelty part of the problem. Failure of certain schemata to participate to the solution is also part of that information and can be used to alter them.

(d) Synthesis of Lisp programs from input output pairs, as reported in [Smith-82],[Biermann-84]. Analysis has to relate the s-expressions in each pair to a set of known basic functions in order to produce a possible trace of the function to be found. Induction will take place on these traces.

(e) Organizing of narrative material in terms of scripts, MOP's and TOP's [Schank-82] can also be presented in this framework. The example is a fragment of text, that is related through analysis to a number of expectation structures. The material that is generalized is the set of expectation failures.

(f) Lets look at the problem of classifying images like Michalski's trains [Michalski-83] (the reader may peek at figure 5.3.1, page 94). Analysis has to relate the drawing to a set of background concepts (geometry, what things can be carried by what shapes of cars, etc.). Several possible representations of each example may result. Only some may contain the adequate descriptors by which the images can be classified.

(g) Consider, finally explanation based learning. The memory contains a theory and analysis consists in building a proof for the example in that theory. Generalization is applied to the proof.

Independently of a representation language, we can define Analysis as the attempt to assemble an example-generating program out of previously stored programs. It results in an explanation for the example based on known terms, and this is the first step for a compression of the ensemble (E, M).

This definition provides support for the intuition that the process of building an explanation can be justified beyond proving an example in a logic Theory. A related argument can be found in [Laird-90] where EBG is extended to functional programs and lambda calculus.

5.2 Explanation Based Learning

We are presenting now explanation based learning (EBL) as a highly formalized instance of analysis. The reader is assumed familiar with EBL. [Mitchell-86],[DeJong-86] and [Minton-89] defined the method and contain good expositions of it.

5.2.1 The Proof As A Code.

Take a formal system S that generates a language L . For every object x in L there exists at least one derivation d , that is a sequence of operations that derives x from the axioms of S . There might be a number of arbitrary choices made in S in order to build d (among applicable rewriting rules, etc.). In fact they are those choices that are made in order to obtain d rather than some other derivation. Let us call the set of those choices c . Then c contains that part of the structure of x that is not determined by the fact that x is an element of L . Using the terms defined in section 3.2, S is a model for x and $c = \text{code}(S, x)$.

Assume now that we have a theory T about some domain D . We are given a number of examples $x_1 \dots x_n$ from D that illustrate some unknown concept $C \subseteq D$. How can we use in the induction of that concept the knowledge that we already have in T ? We derive each x_i from T (that is we use T as a model to parse them) and we extract their codes c_i . If there are any common features among x 's that distinguish them from other elements of D , they must be represented in the particular choices used in T to derive them. Therefore the induction should operate on the codes.

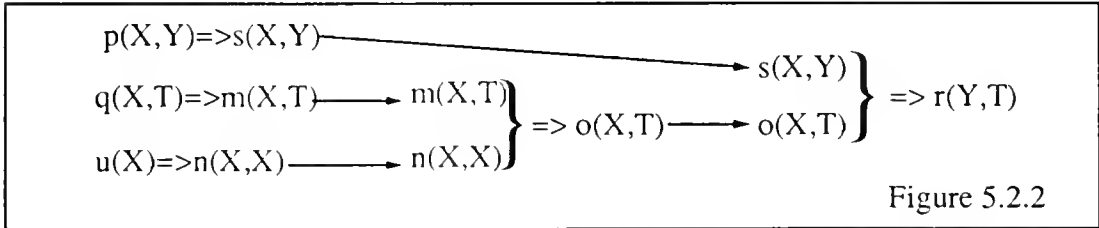
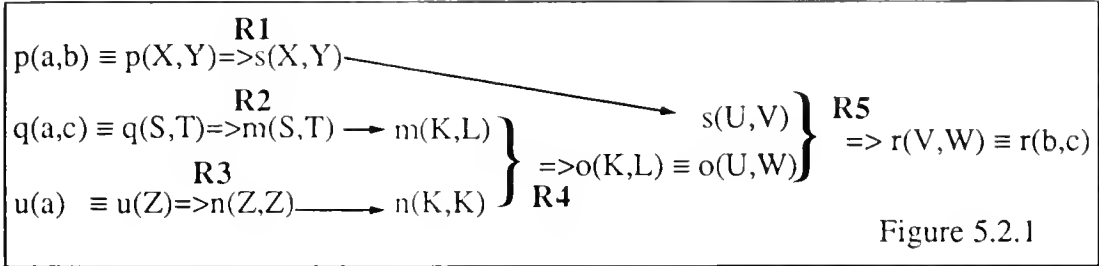
Let T be a first order theory on D and x is a 'fact' from D in our case a grounded Horn clause, say

$$p(a,b) \& q(a,c) \& u(a) \Rightarrow r(b,c).$$

A derivation of x will be a tree structure connecting $r(e)$ at the root with $p(a,b)$ and $q(c,d)$ at the leaves. The particular choices made in T to obtain this derivation will involve the tree structure, the various axioms in the internal tree nodes, and the various substitutions.

That the above ideas can be developed into a foundation for many of the forms of explanation based generalization and learning that are currently pursued (one example, many examples, logical theory, search space, planning). Take for example the method outlined in [DeJong-86]. It would build an explanation structure by attaching copies of uninstantiated rules to the points where they were used in the proof. Then it would retain in a set G the most general unifiers used to couple those rules one to another but not to the grounded parts of the example. Let us say that for the Horn clause

above, the explanation structure is that in figure 5.2.1.



The choices made during the construction of the proof are represented by the tree in figure 5.2.2

together with the substitutions $\sigma = \{ a/X, b/Y, c/T \}$. In fact, assuming that the explanation graph cannot be compressed any further, the structure above can be lambda-abstracted to $h(X,Y,T)$ since the choice of rules(R1,...R5) and the rest of the substitutions are integral part of the graph structure. With this new notation, our proof, that is the *code* for the initial grounded clause is $f(a,b,c)$. Explanation based generalization is based on the remark that only the 'f' part of the code is the structure of the proof and the proof will be valid for any other group of constants.

5.2.2 Generalization of Explanation by Retaining its Structure.

Researchers in EBL found that this technique is not powerful enough in certain situations, and they tried to generalize it. For example EBL was not capturing the recurrence in a plan for building a tower. Therefore it could not generalize the plan to towers of any height. Approaches like [Cohen-88] and [Shavlik-89,90] fall under the case where the explanation graph can be compressed, i.e. it has an identifiable structure. Take as an example the axioms (all variables universally quantified) and the proof below.

{ $A_1(X,Y): p(X,Y) \Rightarrow p(f(X),Y)$,
 $A_2(S,U): p(S,U) \Rightarrow q(S,U)$,
 $A_3(V,W): q(f(V),W) \Rightarrow q(V,g(W))$ }

$$\begin{aligned} p(a,b) \Rightarrow p(f(a),b) \Rightarrow p(f(f(a)),b) \Rightarrow p(f(f(f(a))),b) \Rightarrow \\ q(f(f(f(a))),b) \Rightarrow q(f(f(a)),g(b)) \Rightarrow q(f(a),g(g(b))) \Rightarrow q(a,g(g(g(b)))) \end{aligned}$$

A possible model for the proof structure is

$$h(X,Y,N) = A_1(X,Y)^N A_2(X,Y) A_3(X,Y)^N$$

We conclude that in explanation based learning the theory is the *Memory* element of figure 5.1.1. The *Analysis* process is highly formalized and it constructs proofs. A proof is a *Derivation*. It can be generalized and added to the theory.

But, since the new part of *Example* is already separated, this can be first used as an example for a *similarity based* induction stage. There are already empirical attempts in this direction. For example, several proof trees can be compressed together in order to extract more complex additions to the theory.

If all examples can be proved in the theory, then their common structure can be used to constrain the theory, that is probably too general. The usually claimed purpose of EBL is to obtain *operational* definitions of concepts for which we already have a definition. In our example above, the predicate *r* was defined in terms of *s* and *o*, but after the example presentation it acquired a new definition in terms of *p*, *q*, and *u*, that may be *operational* predicates, that is, computable in the context of the particular application. However, the new definition associates a shorter code to the given example: one rule instead of five!

If some examples cannot be proved, the attempted partial derivations and the places where the *Analysis* process failed can be used to generalize the theory. Some researchers, for example [Mostow-87], [Hall-88], are pursuing the idea of learning from explanation failures.

5.2.3 Specialization of Theory Through Positive Examples.

Traditional approaches to learning needed negative examples in order to specialize a theory. We show now how Explanation-Based Generalization (EBG) can use positive examples to specialize a theory.

Let the theory be a model $m:C \rightarrow E$, that is a function from codes to examples. Endowing a learner with a model *m* as background knowledge is equivalent to constraining its expectations to

$$\{ e \in E \mid (\exists c \in C)(e=m(c)) \} \quad \text{the range of the function } m.$$

Assume that a specific example *e* is presented to the learner. EBG works by gen-

eralizing the proof of e in the given theory. Therefore we assume that e is parsable by m and coded into c .

$$m(c) = e$$

EBG proceeds by generalizing c . Let f be the generalization of c . Since f is more general, it needs some extra information c_0 in order to instantiate to c . Thus we can take f to be a function with values in C , and we can write $c = f(c_0)$. Note that the range of f is a subset $C' \subset C$. The generalization of c induces a generalization of e to $E' = m(C') \subset E$. This amounts to constraining the expectations of the learner from E to E' and the replacement of the initial model m by a more specific one

$$m' = f \otimes m \quad (\text{where } \otimes \text{ denotes the composition of functions}).$$

However not every generalization is acceptable ! Since c represents a computation trace for the machine m , its generalization should have a certain syntax in order to represent a valid computation trace. The function m must be applicable to it in order to produce a representation of a generalization of e . It is possible to generalize c to an f that is no longer in the domain of m . Imagine for example generalizing the graph in figure 5.2.2 by replacing the predicates by variables.

The generalization of c should produce the simplest structure that is still a computation trace for m . That is equivalent to the weakest precondition in goal regression.

Example 5.2.1 Let G_0 be a grammar with the following productions:

- 1) $S \rightarrow \epsilon$
- 2) $S \rightarrow aSa$
- 3) $S \rightarrow bSb$

$$L(G_0) = \{ w\bar{w} \mid w \in (a+b)^n \text{ \& } \bar{w} = \text{reverse}(w) \}$$

The production are numbered in order to use G_0 as a coding model. Every string in $L(G_0)$ will be coded into a string of production numbers, in the order of their applications. Let us examine the string

$$e = aabbbaaabbbaabbbbaabbaaaabbaa.$$

G_0 is a model that parses (analyses) e into the code (explanation) c . That is:

$$c = 22332222332233331 \text{ and } G_0(c) = e.$$

How can we generalize the explanation c ? One way is to retain its structure and discarding its random part. Unfortunately, these concepts are well defined only for infinite objects. A better approximation could be obtained if we generalize several instances together.

For this case, we shall assume a class of models for regularities of c that we can grasp. Therefore, note the following regularity: c is made up of pairs 22 and 33, except for the last character. This enables us to compress it into $c = G_1(c')$ where G_1 is defined as

- 1) $S \rightarrow 1$
- 2) $S \rightarrow 22S$
- 3) $S \rightarrow 33S$

and $c' = 232232331$

Summarizing, the generalization procedure consists in discarding c' and retaining G_1 . The structure of the example is now integrated into the theory G_0 by composition $G_2 = G_1 \otimes G_0$. We operate this composition by noting G_1 is forcing productions 2 and 3 in G_0 to be applied twice in a row. This results in the following G_2

- 1) $S \rightarrow \epsilon$
- 2) $S \rightarrow aaSaa$
- 3) $S \rightarrow bbSbb$

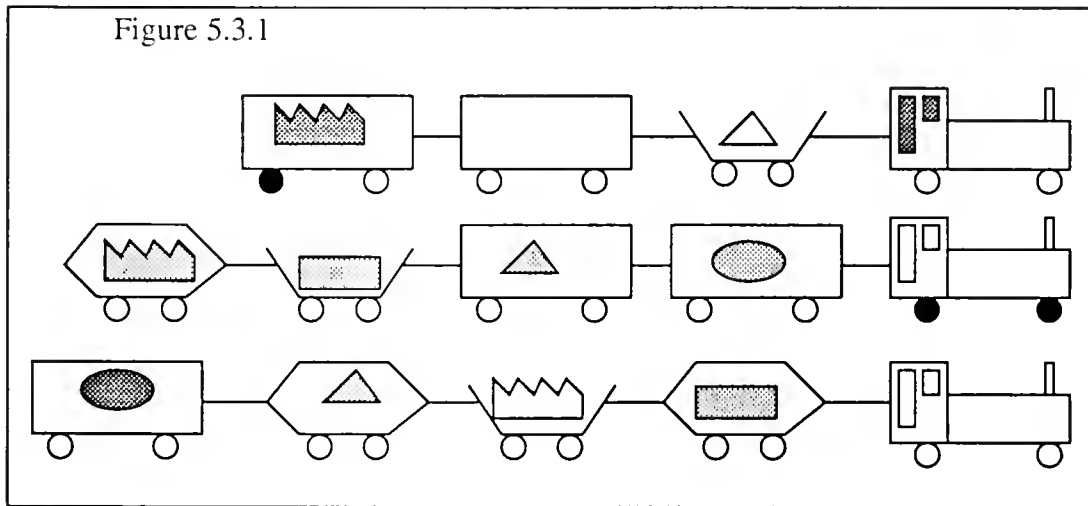
In conclusion G_0 was specialized to G_2 while the set $\{e\}$ was generalized to

$$L(G_2) = \{ w\bar{w} \mid w \in (aa+bb)^n \text{ \& } \bar{w} = \text{reverse}(w) \} \subset L(G_0)$$

5.3 Constructive Induction

Keeping in mind figure 5.1.1 we shall look now at learning models where the *Analysis* part is not as well specified as building a logical proof. Since it has been agreed for a long time that the representation used for *Examples* is not always appropriate, many researchers explored ways to expand the original description language [Michalski-83, Rendell-85,86,89]. Michalski, for example, assumes that the examples are described via some predefined predicates that he calls descriptors. Then the representation alteration amounts to constructing new descriptors, hence the name of constructive induction given to this process.

Example 5.3.1 (Michalsky's Trains)



Let us look at some complex, structured examples like the trains in figure 5.3.1. Assume that the description language is a first order language. Using the predefined predicates, the middle train may be described by the following conjunctive expression:

```
car_of(car1,train2) & car_of(car2,train2) & car_of(car3,train2) &  
car_of(car4,train2) & engine_of(engine,train2) &  
door_color(engine,white) & window_color(engine,white) &  
connected(engine,car1) & connected(car1,car2) & connected(car2,car3) &  
connected(car3,car4) & wheel(engine,w1) & wheel(engine,w2) &  
wheel(car1,w3) & wheel(car1,w4) & wheel(car2,w5) & wheel(car2,w6) &  
wheel(car3,w7) & wheel(car3,w8) & wheel(car4,w9) & wheel(car4,w10) &  
color(w1,black) & color(w2,black) &  
color(w3,white) & color(w4,white) & color(w5,white) & color(w6,white) &  
color(w7,white) & color(w8,white) & color(w9,white) & color(w10,white) &  
contains(car1,obj1) & contains(car2,obj2) &
```

contains(car3,obj3) & contains(car4,obj4) &
shape(car1,rectangle) & shape(car2,rectangle) &
shape(car3,cup) & shape(car4,hexagon) &
closed(car1) & closed(car2) & open(car3) & closed(car4) &
shape(obj1,ellipse) & shape(obj2,triangle) &
shape(obj3,rectangle) & shape(obj4,dented) &
color(obj1,gray) & color(obj2,gray) & color(obj3,gray) & color(obj4,gray)

Although this description does not capture all information conveyed by the drawing in figure 5.3.1, it is far more complex than the attribute-value descriptions on which many similarity based learning algorithms can work. Some of its grounded atomic formulae cannot be interpreted as an *attribute=value* expression. Let us make the following distinction among the features of the above trains:

- 1) Some particularities of the middle train can be represented in the description language itself. However they must be constructed. They are the result of an analysis operated on the initial description. For instance
 $(\forall X)(\forall Y)(\text{car_of}(X,\text{train2}) \ \& \ \text{contain}(X,Y) \Rightarrow \text{color}(Y,\text{gray}))$
- 2) Other particularities, for instance those based on number, can hardly be expressed at all in the description language since they require new predicates (Every car of this train has two wheels).
- 3) A last category includes those particularities that depend on knowledge that is not even implicitly coded in the description. For example, those that depend on the distinction between round and angular shapes, or those that depend on identifying cars that can carry liquids.

[Michalski-83] studied induction from such kind of description and these examples of trains are representative for the inherent difficulties. His programs for diagnosis of soybean diseases, and for classification of microscopic images of cells had to discover such new descriptors. For this purpose, he created a description language APC (Annotated Predicate Calculus) that allows constructions of new descriptors in the context of background knowledge. Some examples are:

- (i) number_of { car | closed(car) } = 3
- (ii) length_of (train) = 5
- (iii) number_of { car | contains(car,obj) & shape(obj,s) & angular(s) } = 3

Note that construction of descriptor (iii) depends on background knowledge on classification of shapes like angular(rectangle) and round(ellipse).

The extensions to the initial description language were not extremely complex

and neither was the background knowledge used. New descriptors were generated, rapidly assessed and pruned by the INDUCE algorithm.

However, the same method applied to more complex situations [Stepp-86], became impractical and more background knowledge had to be postulated to help choose among possible constructions. For example, given a large set of such toy trains, many valid classifications are possible. A general classification goal (or preference for certain classification criteria) and knowledge on subgoals dependencies had to be assumed.



Now let us step back and attempt to formalize this approach. A simple sketch of this method for constructing new descriptors is the following. There is an original description language L_0 and a set F of operators (constructors) that can be applied to L_0 expressions to create new descriptions. These alterations are in fact equivalent to specifying a more complex language L_1 that contains L_0 and all the permitted extensions of it:

- (i) $L_0 \subseteq L_1$
- (ii) $(\forall e \in L_0)(\forall f \in F) f(e) \in L_1$

In order to make this apparently simple definition work, we need to answer a few questions.

(a) When are alterations needed and which is the desired one? A typical answer to this was to assume an induction algorithm that is tried on the altered expressions. For instance, if a reasonable size decision tree cannot be induced in terms of existing descriptors, then new ones have to be constructed.

(b) Do all constructions make sense? When can an operator be applied? The conditions of applicability of operator are part of the background knowledge. In the next section we shall give an example of construction guided by expression types.

(c) Are there particular properties of the domain, or formal properties of the operators, that determine the new descriptors to be equivalent or subsumed to one another? For example, in figure 5.3.1, the descriptors *the number of cars* and *half the number of wheels* are equivalent for this domain. An answer to this question may simplify enormously the task of construction. One way to approach this is to provide a complete axiomatization for the semantics of L_1 . A different way is the one used by discovery programs such as AM [Lenat-78,83,84] and

Cyrano [Haase-87,88]. The semantics of new constructions is studied empirically, on examples.

Looking for common patterns by continuously reevaluating the examples in terms of known concepts is the essential process in constructive induction. Submitting examples to a number of alterations is an instance of the *Analysis* process discussed in section 5.1. The purpose, again, is to find better descriptions that can be properly compared to each other and to the background knowledge, and through this, to make the Known and the New explicit.

A powerful model for constructive induction should be a model for the way we operate with natural language concepts. The space of possible constructions should be large enough to allow for relating the example to the background knowledge in more than one way. Take for example Hofstadter's exercise of constructing analogues for the concept of *The first lady* for so many contexts [Hofstadter-85]. Unfortunately, like so many examples from natural language processing, Hofstadter's assumes a vast, unspecified universe of background knowledge. A better context for computational models is offered by the geometric domain used in [Bongard-75]. This will be the subject of the next section.

5.4 Bongard Problems

We are given a simple geometric drawing as an example of some unknown concept. Two possible descriptions of this drawing, d1 and d2 are presented to us. The first is a digitized image. The second is the following string

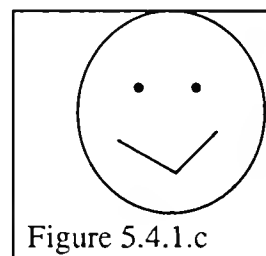
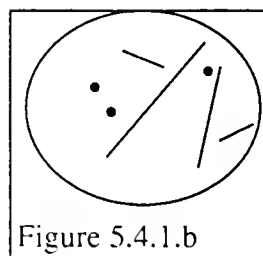
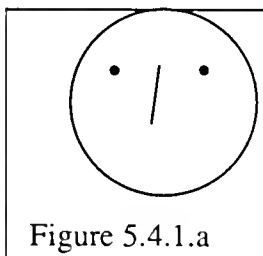
```
{ circle( center:point(50,50), radius:30 ), point(35,60), point(65,58),  
  line-segment( point(50,60), point(48,40 ) ) }
```

Analysis has to identify first the relevant description for perception of this example.

Description d1 is appropriate if the concept illustrated by the example has a short description in terms of the distributions of bits in the digitized image. For instance, d1 may only illustrate the concept *a sparse distribution of 1's*.

Description d2 is appropriate if the illustrated concept has a short description in terms of the geometrical components: circle, line_segment and points. Related concepts like the inside and outside regions delimited by the circle may be relevant too. For instance d1 may stand for *all points and line_segments are inside the region delimited by the curve*.

Another description d3, at a different conceptual level may be relevant too (figure 5.4.1.a). The illustrated concept may be *a sketchy human face*.

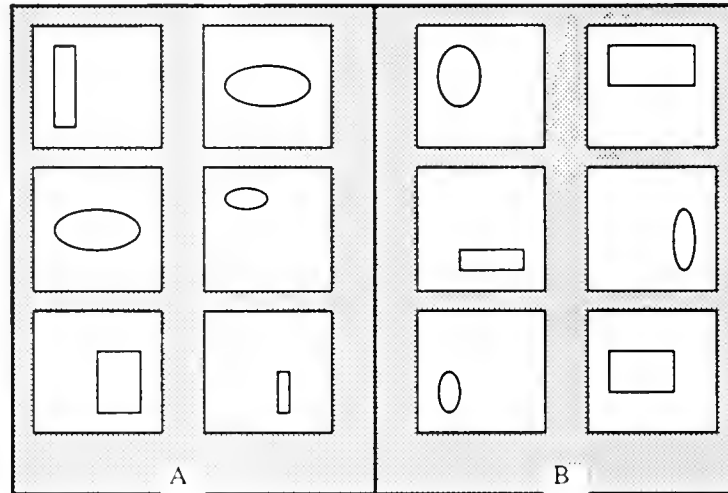


Assuming a *Memory* where all basic concepts used above are defined, there are several short descriptions (parsings) for the example, and the appropriate one will be selected only by the context provided by further examples. Figure 5.4.1.b would support the level suggested by d2, while figure 5.4.1.c would support the one suggested by d3. This is the framework to studied in [Bongard-75] and here is one of the problems proposed there.

Figure 5.4.2 contains the problem no. 13 from the series of a hundred presented in the appendix of the book mentioned above. Two sets of figures, six of each, are given as representative for two classes A and B. Each example is a square containing an

image. The problem consists in finding the best way to discriminate between further instances of the two classes.

Figure 5.4.2



Although the images are quite complex, the discrimination is usually simple. Here all images are ellipses or rectangles and all have a main axis. To discriminate note that in class A rectangles have a vertical main axis and ellipses a horizontal one. Therefore there are two attributes (shape and posture) essential to the description of both classes and the discrimination can be expressed as a simple boolean function :

Class A(X) : (rectangle(X) & vertical(X)) \vee (ellipse(X) & horizontal(X))

Class B(X) : (rectangle(X) & horizontal(X)) \vee (ellipse(X) and vertical(X))

Many times, though, discrimination can be done by the value of a single attribute that is common to both classes. Since the shape (rectangle/ellipse) and the existence and direction of the main axis are not likely elements of an initial description language for these instances, they are attributes that must be discovered and constructed into the representation. For some problems in the series, these relevant attributes are quite complex when expressed in terms of a low level description. And yet humans identify them in seconds. This is due, to a large extent, to the fact that Bongard problems are strongly biased towards common human concepts and towards certain perceptual groupings that the human eye can operate in a preattentive mode.

Typical concepts that the solver is expected to identify are: convexity, concavity, trees, polygons, containment of regions, body vs. tentacles or lobes, letter shapes. Perceptual groupings like proximity, colinearity, figure vs. background, contour, bimodal texture are essential to highlighting certain features. That is why the modifications of the initial representation, whatever that may be, must be guided by a large amount of background knowledge.

How should this background knowledge be organized? First, it should contain a dictionary of shapes, and visual routines able to recognize them. Research in computer vision perfected methods to make salient similarity of shapes through rotation translation, scaling, continuous deformation etc. An ellipse in the image, for instance, may be parsed as one of the following:

- (a) Ellipse(focal_points, size)
- (b) Scaling(Circle(center,radius),direction,size)

All (reasonably short) parsings should be considered since anyone may emerge as essential. Relationships among shapes, rather than shapes alone, may provide the relevant attributes. Unfortunately they cannot be simply listed in the dictionary next to each shape since they may be mediated by other shape. For example *the square and the circle are on the same side of the large diagonal line segment* relates two shapes via a region defined by the position of a third. As with Tiles-land elements already present in a description may suggest additional components through associative retrieval. The presence of the line segment may suggests a two region partition and may prompt a re-evaluation of the other component positions in terms of it.

The solution to a Bongard problem is a *discriminating concept*. However this concept accepts a short description only if it is defined in terms of another set of concepts that are common to instances of both classes. In figure 5.2.4 the discriminator is based on *shape* and *main axis*. We shall call this basis for comparison the *underlying concepts*. In most problems they are the real target. Once they are identified, the discrimination becomes very simple. Since they can be named, it is assumed that the underlying concepts already exists in Memory. They only have to be identified as the model that can parse all instances of both classes into short codes. Therefore, if m stands for the set of underlying concepts, the compression operated by m on all instances is the heuristic that can guide the search:

$$\text{compression}(m) = \sum_{i=1,6} |\text{code}(m, \text{instance}_{1,i})| + \sum_{i=1,6} |\text{code}(m, \text{instance}_{2,i})|$$

We shall now formulate a simplified version of Bongard problems where *the constructive induction can be guided by a theory of types*. The task is to find a set of constructive operators with certain desirable properties. We shall take all instances to be described in some initial representation R_0 . We shall define operators that act on descriptions and generate new descriptors or attributes.

Assume that we attach a type to each descriptor. Also for each constructive operator we specify the type on which it applies and the possible types of the result. Therefore, if one descriptor is of type A and the operator o maps type A into type B, the resulting descriptor will be of type B. In fact, instead of associating two types with

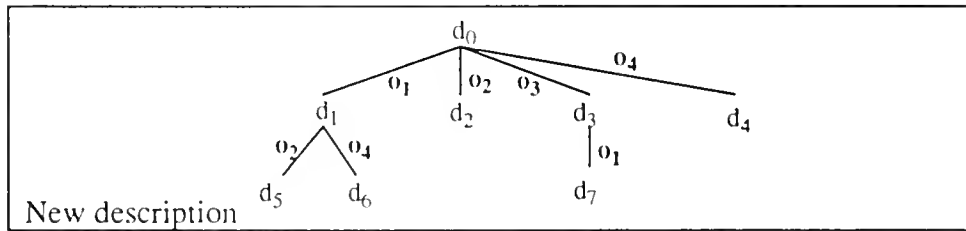
an operator, we shall provide a way to compute an expression for the resulting type from the one for the original type.

$$d:A \ \& \ o:A \rightarrow B \ \& \ d' = o(d) \Rightarrow d':B$$

Since several operators may apply on the same descriptor, the possible alterations will be structured as trees with the initial description at the root, the other nodes labeled by the new descriptors and the branches labeled by operators. (figure 5.4.3) We shall use the following terms.

A computation is a string of operators. Any computation applied on the original description is a new attribute and the result of that computation is the attribute's value. A descriptor is a pair $\langle \text{attribute}, \text{value} \rangle$.

4.3



Also we assume the existence of a learning algorithm L , that given the descriptions of 4 instances of class A and 4 instances of class B , attempts to build a discriminator as a simple boolean function of descriptors. For example, in figure 5.4.3, a discriminator might be $(d_5 \ \& \ d_7)$ or, in terms of operators, $((o_2(o_1(d_0))=v_5) \ \& \ (o_1(o_3(d_0))=v_7))$. In this case d_5 and d_7 are the underlying concepts.

Let us raise two issues about constructed descriptors:

(i) We assume that a descriptor describes! However it is possible that, certain combinations of operators realize constant functions. If $(\exists v)((\forall d)(o(d) = v))$ then the descriptor $o(d)=v$ does not carry any information about d . Therefore, it is no use to construct new descriptors based on o . Such operators should be detected.

(ii) There is a natural limit to the complexity of new descriptors. In chapter 3, a model m needed a positive compression factor in order to capture a regularity. Here, a new description based on the constructor o may be more complex than the initial d_0 . This is acceptable to a certain extent, since the compression is not applied only to the string of examples, but to the pair $[\text{examples}, BK]$. Thus, we can tolerate a complex construction as long as the new description allows, given enough examples, a compressed generalization.

Referring again to figure 5.4.2, such a limit should discourage candidate concepts like *the number of common prime dividers between the size of the surface inside the curve and the size of the maximum surface delimited by two parallel tangents to the curve and the image frame*.

For the first issue one may attempt to provide axioms that allow inference of such cases. For the second issue, we shall take advantage of the fact that, in Bongard problems, the number of examples is finite. Let $e_1 \dots e_{12}$ be the examples in their initial representation and $e_1' \dots e_{12}'$ be the examples in a new representation constructed with the operators o_1, o_2 . It is possible that

$$(1) \quad \Sigma \text{plex}(o_i) + \Sigma |e_j'| > \Sigma \text{plex}(e_j)$$

but the new representation allows the induction of a model m that can code $e_j' = m(c_j)$ such that

$$(2) \quad \Sigma \text{plex}(o_i) + \text{plex}(m) + \Sigma |c_j| < \Sigma \text{plex}(e_j)$$

For (2) to hold, we need at least $\Sigma \text{plex}(o_i) < \Sigma \text{plex}(e_j)$ which gives us an upper bound for the complexity of reasonable constructions.

In order to work out an entire example, we have to further constrain the geometric universe of Bongard problems. Figure 5.4.4. illustrates a smaller domain. The instances are one dimensional images, namely sets of points on a line segment. The points are marked by either an x or a $+$.

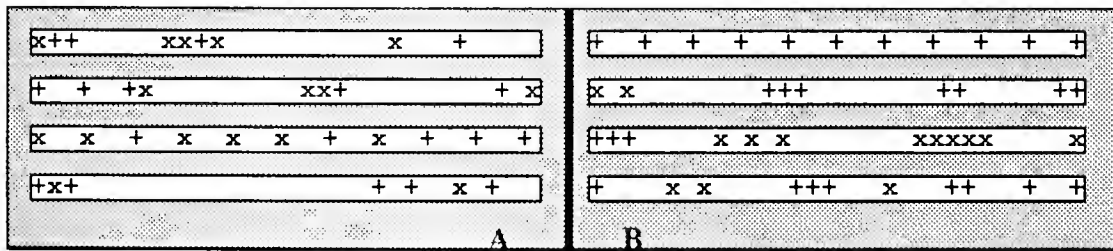


Figure 5.4.4

Expressed in the initial representation every example will be a set of points. We need to define primitive types for our type theory. They will be **Point** and **N** (natural numbers). Besides, for every type **Type** we shall have the composite **setof Type**. The constructive operators are the following:

card : setof Type \rightarrow N

This simply takes the cardinal of a set of any type, and yields a natural number.

setof card : setof setof Type \rightarrow setof N defined as
setof card Y = { card(x) | x \in Y }

The card operator can be prefixed by one or more occurrences of setof.

face : setof Point \rightarrow setof setof Point

This partitions the set into one or two sets based on their markings so that one subset will contain only x's or only +'s. For example, applied on the upper left instance in figure 5.4.4, face returns a set of two sets: one of five x points and one of four + points.

cluster : setof Point \rightarrow setof setof Point

This will partition the set into clusters based on the relative distance, so points close to each other will get into the same set. For example, the second instance of class A would be partitioned into 3 sets, but the third instance would be partitioned only into one set since the points are equidistant. This operator attempts to simulate a perceptual grouping based on proximity.

Since type expressions can become complex by recursion, we also define a rule schema for application of face and cluster to the elements of a set. Let o be one of these two operators.

$o:\text{Type1} \rightarrow \text{Type2} \ \& \ r:\text{setof Type1} \Rightarrow o(r):\text{setof Type2}$

We also assume that we can simple discriminating boolean functions based on the predicates

$x = y$ where x:Type, y:Type

$x \in y$ where x:Type, y:setof Type

All examples in figure 5.4.4 are of type e:setof Point. One discriminating function is

$\text{setof card} (\text{face} (\text{cluster} (e))) = \{ 1 \}$

This is false for examples of class A and true for examples of class B. Its English equivalent is: *Each cluster in class B example has points marked with one face.*

6. Discussion and Research Directions

Let us resume now the main ideas of each chapter. We shall restate the results informally and outline possible continuations.

Chapter 2. Among the proposed themes, the idea of algorithmic complexity looms large. That is why we started by presenting the Kolmogorov complexity, and some related concepts from the algorithmic information theory. Koppel's distinction between structure and randomness is equally important and it allowed us to formalize regularity. It is also the key in defining natural spaces, that is, those we wanted to be concerned with. They are, essentially, those processes whose manifestations can be simulated by a finite program that reads a random string.

We presented a derivation of the minimum description length principle from the Bayesian approach to learning. Applications of this principle need a computable approximation for the algorithmic complexity. We presented Hart's approximation based on grammatical derivation and suggested a possible improvement of it.

We described Solomonoff's promising but sketchy strategy for unifying problem solving and learning. We attempted to clarify some details. Computation of new operator probabilities, based on compression of two strings, seems to have some inherent difficulties.

Nevertheless presentation of the learning phase as a second problem solving phase allowed us to identify improvement in learning skills as a necessary part of learning. It also permitted us to distinguish between the physics and the mathematics components of learning. This distinction will become apparent again in constructive induction. A complete computational theory of learning and problem solving is still an open subject.

We could now ask the question: Are all forms of learning based on compression? When we see learning as a process of discovering regularity, the answer seems to be yes. There is, however, the phenomenon of knowledge compilation, that is also classified as learning. It transforms a known short program into an equivalent longer one to increase its execution speed.

Chapter 3. Here we used the MDL principle to formalize regularity as any model that achieves compression. We then distinguished between complete and partial models.

We investigated several approaches to capturing regularities. We argued that a measure for compression offers good guidance against over generalization in induction from positive examples. We discuss model refinement and convergence.

We established sufficient conditions for an incremental algorithm based on refinement of models. We showed the following:

- ◆ If the data sequence has a stable finite structure, the refinement algorithm converges.
- ◆ There exists a point beyond which dropping less performant hypotheses does not affect convergence.
- ◆ The compressibility achieved by a model directly affects its ability to predict the continuation of the sequence.

Then, we introduced Finite Memory Models (FMM), as finite automata (FA) with a vector representation for states. We established several things about them:

- ◆ Although their recognizing power does not exceed that of FA's, they can express more regularities. For example, they can express the presence of isomorphic subgraphs in a FA transition graph.
- ◆ They can also express an FA as an ensemble of loosely coupled, simpler FA, and, with this, they proved adequate for the microworlds in section 4.3.
- ◆ They are incrementally inducible by the algorithm mentioned above, and also closed under composition, thus adequate for stepwise induction.

We focussed then on a direction of research on hierarchical structures started in [Caianiello-71] and continued in [Caianiello-89]. We defined Libraries as collections of partial models. Libraries are the basis for accumulation of partial knowledge. We showed them to generalize syllabaries and classifications introduced by P. Caianiello as regularities of text, as well as certain types of clustering.

We investigated induction of libraries. We presented the process of using a library for parsing as a set of parallel computations that alternate among three phases: confusion, recognition of an identifying sequence, prediction. We advanced the following claims about libraries.

- ◆ The components of libraries, routines, determine the fragmentation of the input data stream.
- ◆ For highest compression, routines should not contain common information; they should be algorithmically independent of each other.

- ♦ In order to maintain its compressibility a library will have to expand some routines, or introduce new routines to parse new data. The decision depends on the algorithmic similarity between the new data and the existing routines.
- ♦ It will also have to replace a routine by simpler ones when the data fragments parsed by it are no longer uniformly distributed; or replace several routines by a single one. These operations have their equivalents in incremental conceptual clustering algorithms.

We defined then the concept of basic regularities as an algorithmically independent components of structures. We showed how basic regularities can be identified by local maxima of compressibility as a function of model complexity.

The above concepts permitted us to define stepwise induction as a general strategy for avoiding the intractability of certain induction problems. Many natural structures lend themselves to this approach since they are hierarchical and their level complexities are bounded.

We worked out an example (3.6.2) of compression of a geometrical image with Library models. We illustrated both the identification of basic regularities and the stepwise induction of the hierarchical structure.

We presented the results of a program (example 3.6.3) that incrementally induces complete automata models. We applied the program on a hierarchical structure and showed the emergence of the hierarchy levels as local maxima of compressibility expressed as a function of complexity.

Stepwise induction depends on classes of models that can code the data into structures analyzable by models in the same class. Better characterization of these classes is an interesting open problem. Could primitive recursive classes of function be approached by stepwise induction ?

Stepwise induction seems to be related to some ideas of [Minsky-85] on using a second learning agent to supervise a first one that analyzes external data. After building a high level of confidence on the stability of the first agent's findings, the second would be introduced to look for second order regularities .

Chapter 4. Some AI researchers invoked every source, from Heidegger to memories of how they handle the door when their hands are loaded with bags from the supermarket, to argue that intelligence cannot be understood as a disembodied computation. (see, for example [Brooks-85]).

That is why we focussed here on algorithms endowed with a simulated body that

engage in active learning. They accept an infinite string produced by an unknown generator but they can control it, to a certain extent, by acting on the generator. They model the situation of an agent exploring a finite, deterministic environment.

We showed that some active learning problems can be reduced to induction on the sensory-motor stream, that is, an alternation of actions and sensorial images. From a psychological perspective these investigations could offer support for a constructive view of cognitive development. They open a possibility to provide a basis for language learning.

The main objective here was to find what kinds of environment regularities can be induced in different contexts. We assumed that the agent executes a random sequence of actions. Thus, the environment regularities determined the structure part (see section 2.3) of the sensory-motor stream. We studied active learning in three, increasingly complex, situations.

(i) We started with an agent that feeds input symbols to, and reads the output of a Moore automaton. Active learning offers a natural way to build new descriptions by aggregating actions and sensory images. We showed that two sets of characterizations for a hidden state result from here:

- ◆ the first set contains sensory-motor sequences obtained by leaving the state
- ◆ the second set contains sensory-motor sequences obtained in reaching the state.

We introduced the second kind as *id_sequences* and we proved their existence for Moore automata. Following these characterizations, we provided two incremental algorithms:

- ◆ one refines of an initial model to maintain its determinism
- ◆ the second accumulates partial regularities, and alternates between the three modes: regularity recognition, prediction based on the recognized regularity, and confusion.

(ii) Then we defined a class of microworlds of loosely coupled automata. We illustrated, with several examples, how several simple physical universes can be modelled by these worlds, and how FMM's, introduced in the previous chapter, provide a versatile way to capture their regularities.

Such a world can also be represented as a Moore automaton. In this context we investigated regularities induced by two kinds of substructure equivalence.

- ♦ isomorphism of disjoint fragments in the transition graph.
- ♦ equivalence of tests (defined in [Rivest-87b])

We argued that, under certain conditions, induction in this space operates a construction of reality, that is, it creates models for both a space map and independent objects.

On a specific example we illustrated several representations: Moore automaton, FMM, library of automata, partial id_sequences (defined here). We are working on a library induction algorithm applied to this example.

(iii) Expanding on [Drescher-86], we considered an agent endowed with a complex eye and a hand, pushing flat objects on a tiled surface. Objects and hand were perceived as tile markings.

We showed that complex relationships between tiles markings and actions can be captured in relation schemata by a simple form of statistical induction combined with lambda abstraction.

We then gave an outline for acquisition of complex computations based on several relation schemata. This scenario is related to the library induction problem where several routines have to be combined into one computation.

Tiles-land also illustrates, to a certain extent, the concept of internalization: stable transitions perceived in the outside world become internal computation steps. A interesting, related subject is the acquisition of new computation and representation devices through acquisition of language. For example, heard and memorized accounts about external events may participate as steps in an internal computation. This way the agent imports from outside (by simulation) a computational power that resides in language, or in the perceived physical universe. (See the arguments about using the world itself to compute [Chapman-87]. Also the arguments about meta-representations in [Lloyd-89].)

Research in common sense reasoning suggests that stability regions in the phase space are good candidate regularities to acquire. To what extent they can be captured, from passive perception of movement or through interaction, seems worthwhile to explore.

Chapter 5. Here we returned to the question: how are models refined in order to incorporate new data? We defined analysis as the process of separating the part that a new piece of data shares with the background knowledge from the part that is entirely new. We related it to hypotheses refinement. We presented several instances of analy-

sis from different areas of learning.

We discussed then two extreme cases of analysis. The first is highly formalized and occurs in Explanation-Based Generalization. The second has many ad hoc implementations and occurs in constructive induction.

The concepts of model and code cast new light on EBG. They allowed us the following:

- ◆ to establish its generality beyond first order logic (on different grounds than the similar generalizations in [Laird-90])
- ◆ to propose new ways to treat proof generalization based on capturing structure
- ◆ to understand it as a form of focussing (specialization) of too general hypotheses (see example 5.2.1)

We defined the process of constructive change of representation, or constructive induction as a search through a space of representations. We related this search to the exploration of the formal properties of constructive operators, that we called, in section 2.5, the mathematics of learning.

We illustrated the issues with examples of Michalski trains and Bongard classification problems. For a simple class of Bongard problems we sketched a theory of types that constraints the space of possible constructions. We illustrated in this context the issue of too complex descriptors, and the issue of empty descriptors.

Conclusion. The problems discussed here are necessary aspects of a computational model of natural learning. The algorithmic information theory and the concepts of partial model and basic regularity were instrumental in presenting a unify view of these aspects. Active agents are the proper context for understanding learning in live beings and to integrate problem solving and acquisition of language.

BIBLIOGRAPHY

[Angluin-78] Angluin D. - Learning Regular Sets from Queries and Counterexamples, *Information and Computation* 75 (November 1987) pp. 87-106

[Angluin-79] Angluin D. - Finding Patterns Common to a Set of Strings, *Research Report at Univ. of California at Santa Barbara*, 6/1979

[Angluin-80] Angluin D. - Inductive Inference of Formal Languages From Positive Data, *Information and Control* 45 (1980), pp 117-135

[Angluin-83] Angluin, D. & Smith, C.H. - Inductive Inference: Theory and Methods, *ACM Computing Surveys* 15 (1983) pp. 237-269

[Bennett-85] Bennett, Ch.H. - Dissipation, Information, Computational Complexity and the Definition of Organization, in Pines D. (ed.) : *Emerging Syntheses in Science* (pp 297) Santa Fe Institute, New Mexico (1985) (also Addison Wesley)

[Bennett-88] Bennett, Ch.H. - Logical depth and Physical complexity, in Herken, R. (ed.): *The Universal Turing Machine - a Half-Century Survey* Oxford University Press 1988

[Berwick-86] Berwick R.C. - Learning From Positive-Only Examples: The Subset Principle and Three Cases of Study, in Michalski, Carbonell, Mitchell (eds.): *Machine Learning* (vol2), Morgan Kaufmann 1986

[Biermann-84] Biermann, A.W. & Guiho, G. & Kondratoff, Y. - Automatic Program Construction Techniques, *Macmillan Publ. Co.* 1984

[Blum-75] Blum L. & Blum M. - Towards a Mathematical Theory of Inductive Inference, *Information and Control*, 28 (1975) pp 125-155

[Bongard-75] Bongard - Pattern Recognition, *Basic Books*, 1975

[Botta-90] Botta, A. - Induction of Moore Automata, (*submitted to Machine Learning*, Kluwer Academic Publ.)

[Brooks-86] Brooks R. A. - Achieving Artificial Intelligence Through Building Robots, *A.I Memo* 899, *M.I.T A.I. Lab.* (May 1986)

[Caianiello-71] Caianiello E.R., Capocelli R.M. - On Form and Language: The Procrustes Algorithm for Feature Extraction, *Kybernetik* 8, pp 223-233 (1971) Springer Verlag

[Caianiello-89] Caianiello P. - Learning as Evolution of Representation, *New York University Ph.D. Thesis* (1989)

[Chaitin-75] Chaitin G.J. - A Theory of Program Size Formally Identical to Information Theory, *Journal of the A.C.M.* Vol 22 No.3 (July 1975) pp 329

[Chaitin-77] Chaitin G.J. - Algorithmic Information Theory, *IBM Journal of Research and Development* (July 1977) pp 350

[Chaitin-78] Chaitin G.J. - Toward a mathematical Definition of "Life", in Levine R.D. & Tribus M. : *The Maximum Entropy Formalism*, M.I.T. Press (1978)

[Chapman-87] Chapman D. & Agre P.E. - Abstract reasoning as Emergent From Concrete Activity, *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Morgan Kaufmann Publ. 1987

[Chomsky-79] Chomsky N. - On Cognitive Structures and Their Development, in Piatelli-Palmarini M. (ed.) *Language and Learning: The Debate Between J. Piaget and N. Chomsky*, Routledge and Kegan Paul, London 1979

[Cohen-88] Cohen W.W. - Generalizing Number and Learning from Multiple Examples in Explanation Based Learning, *Proceedings of the International Conference on Machine Learning* 1988, pp 256

[Cover-85] Cover T. - Kolmogorov Complexity, data Compressing and Inference, in Skwirzinski J.K. (ed.) *The Impact of Processing techniques on Communications*, Martinus Nijhoff Publ. 1985

[DeJong-86] DeJong G.F. & Mooney R. - Explanation-based Learning: An Alternative View, *Machine Learning* 1 (1986) pp 145-176

[Drescher-86] Drescher G.L. - Genetic A.I. - Translating Piaget into LISP, *M.I.T. A.I. Lab Memo* 890 (1986)

[Fodor-75] Fodor J.A. - The Language of Thought, *Crowell Publ.* 1975

[Gennari-89] Gennari J.H. & Langley P. & Fisher D. - Model of Incremental Concept Formation, *Artificial Intelligence* 40 (1989) pp 11

[Gentner-83] Gentner D. - Structure Mapping: A Theoretical framework for Analogy, *Cognitive Science* vol 7, No 2 (1983) pp 155-170

[Gerwin-74] Gerwin D.G. - Information Processing, Data Inferences, and Scientific Generalization, *Behavioral Sciences* 19 (1974) pp 314-325

[Gold-72] Gold, E.M. - System Identification via State Characterization, *Automatica* 8 (1972) pp. 621-636

[Gold-78] Gold, E.M. - Language Identification in the Limit, *Information and Control* 37 (1978) pp 302-320

[Gould-87] Gould, J.L. & Marler, P. - Learning by Instinct, *Scientific American* (Jan 1987) pp 74-85

[Haase-86] Haase, K.W.Jr. - Discovery Systems, *AI Memo* 898 (1986) MIT AIL

[Haase-87] Haase, K.W.Jr - TYPICAL : A Knowledge Representation System for Automated Discovery and Inference, *Technical Report* 988 (1987) MIT AIL

[Haken-76] Haken, H. - Synergetics - An Introduction, *Springer-Verlag* 1976

[Hall-88] Hall R. - Learning by Failing to Explain: Using Partial Explanations to Learn in Incomplete or Intractable Domains, *Machine Learning Vol. 3 No. 1* (1988)

[Hart-87] Hart G.W. - Minimum Information Estimation of Structure, *Ph.D. Thesis* (1987) MIT LIDS-TH-1664

[Hinton-89] Hinton G.E. - Connectionist Learning Procedures, *Artificial Intelligence* 40 (1989) pp 185

[Hofstadter-85] Hofstadter D. - Metamagical Themes, *Basic Books*, 1985

[Holder-89] Holder L.B. - Empirical Substructure Discovery, *Proceedings of the 6th Intl. Machine Learning Workshop* (1989) pp 133

[Kanerva-87] Kanerva, P. - Sparse Distributed Memory, *MIT Press* (1987)

[Koppel-88] Koppel M. - Structure, in *Herken, R. (ed.): The Universal Turing Machine - a Half-Century Survey* Oxford University Press 1988

[Laird-87] Laird Ph. - Learning From Good Data and Bad, *Ph.D. Thesis*, YALEU/DCS/TR-551, Yale University 1987

[Laird-90] Laird Ph. & Gamble E. - Extending EBG to Term-Rewriting Systems, *Proceedings of the 8th National Conference on AI, AAAI Press / MIT Press*

(1990), pp 929

[Langley-87] Langley P. & Simon A. & Bradshaw G.L. & Zytkow J.M. - Scientific Discovery - Computational Explorations of the Creative Process, *M.I.T. Press* 1987

[Langley-89] Langley P. & Zytkow J.M. - Data-driven Approaches to Empirical Discovery, *Artificial Intelligence* 40 (1989) pp 283

[Lempel-76] Lempel, A. & Ziv, J. - On the Complexity of Finite Sequences, *IEEE Transactions in Information Theory*, Vol IT-22, No 1 (January 1976) pp 75

[Lempel-77] Lempel, A. & Ziv, J. - A Universal Algorithm for Sequential Data Compression, *IEEE Transactions in Information Theory*, Vol IT-23, No 3 (May 1977) pp 377

[Lempel-78] Lempel, A. & Ziv, J. - Compression of Individual Sequences via Variable-Rate Coding, *IEEE Transactions in Information Theory*, Vol IT-24, No 5 (September 1978) pp 530

[Lenat-78] Lenat D.B. - The Ubiquity of Discovery, *Artificial Intelligence* vol 9 (1978) pp. 257-285

[Lenat-83] Lenat D.B. - Eurisko: A Program That Learns New Heuristics and Domain Concepts, *Artificial Intelligence* vol 21 (1983) pp. 61-98

[Lenat-84] Lenat D.B. & Brown J.S. - Why AM and Eurisko Appear to Work, *Artificial Intelligence* vol 23 (1984) pp. 269-294

[Lloyd-89] Lloyd D.E. - Simple Minds, *M.I.T. Press*, 1989

[Lloyd-90] Lloyd S. - The Calculus of Intricacy: Can the Complexity of a Forest Be Compared with That of Finnegans Wake? *The Sciences*, Sept-Oct. 1990, *The New York Academy of Sciences* 1990, pp. 38

[Maturana-86] Maturana H.R. - The Biological Foundations of Self-Consciousness and The Physical Domain of Existence, in E.R. Caianiello (ed.): *The Physics of Cognitive Processes*, (proceedings of a 1986 conference in Amalfi), *World Scientific* 1986, pp 324-379

[Michalski-83] Michalski R.S. - A Theory and Methodology of Inductive Learning, in Michalski, Carbonell, Mitchell (eds.): *Machine Learning* (vol1), *Tioga Publ.* 1983

[Minsky-85] Minsky M. - The Society of Mind, *Simon and Schuster* (1985)

[Minton-89] Minton S. & Carbonell J.G. & Knoblock C.A. & Kuokka D.R. & Etzioni O. & Gil Y. - Explanation-Based Learning: A Problem Solving Perspective, *Artificial Intelligence* 40 (1989) pp

[Mitchell-78] Mitchell T.M. - Version Spaces: An approach To Concept Learning, *Ph.D. Thesis, Stanford University*, 1978

[Mitchell-83] Mitchell T.M. & Utgoff P.E. & Nudel B. & Banerji R. - Learning By Experimentation: Acquiring And Refining Problem Solving Heuristics, in *Michalski, Carbonell, Mitchell (eds.): Machine Learning (vol I), Tioga Publ.* 1983

[Mitchell-86] Mitchell T. & Keller R. & Kedar-Cabelli S. - Explanation Based Learning - A Unifying View, *Machine Learning* 1 (1986)

[Mostow-87] Mostow J. & Bhatnagar N. - Failsafe - A Floor Planner That Uses EBG To Learn From Its Failures, *Proceedings of the 10th IJCAI, Morgan Kaufmann* 1987, pp 249

[Paturi-89] Paturi, R. & Rajasekaran, S. & Reif, J. - The Light Bulb problem, *Proceedings of the 2nd workshop on Computational Learning Theory*, (1989) pp. 261

[Piaget-54] Piaget J. - The Construction of Reality in the Child, *Basic Books* (1954)

[Piaget-69] Piaget J. & Inhelder B. - The Psychology of the Child, *Basic Books* (1969)

[Piaget-79] Piaget J. - Language Within Cognition, in *Piatelli-Palmarini M. (ed.) Language and Learning: The Debate Between J. Piaget and N. Chomsky*, *Routledge and Kegan Paul*, London 1979

[Piatelli-Palmarini-79] Piatelli-Palmarini M. (ed.) - Language and Learning: The Debate Between Jean Piaget and Noam Chomsky, *Routledge and Kegan Paul*, London 1979

[Powers-89] Powers D.M.W. & Turk C.C.R. - Machine Learning of Natural Language, *Springer-Verlag* 1989

[Quinlan-89] Quinlan & Rivest, R. - Inferring Decision Trees Using the Minimum Description Length Principle, *Information and Computation* 80 (1989) pp. 227

[Rendell-85] Rendell L.A. - Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search, *Proceedings of 9th IJCAI Morgan Kaufmann 1985* pp. 650-658

[Rendell-86] Rendell L.A. - A General Framework for Induction and a Study of Selective Induction, *Machine Learning 1* pp.177-226 (1986) *Kluwer Academic Publishers*

[Rendell-89] Rendell L. - Comparing Systems and Analyzing Functions to Improve Constructive Induction, *Proceedings of the 6th Intl. Machine Learning Workshop (1989)* pp. 461

[Rissanen-83] Rissanen, J. - A Universal Data Compression System, *IEEE Transactions on Information Theory IT-29* (1983) pp. 656-664

[Rissanen-86] Rissanen, J. - Stochastic Complexity and Modeling, *The Annals of Statistics 14* (1986) pp. 1080

[Rivest-87a] Rivest, R.L. - Lecture Notes in Machine Learning, *M.I.T. C.S. Lab.* 1987

[Rivest-87b] Rivest, R.L. & Schapire R.E. - Diversity-Based Inference of Finite Automata, *Proceedings of The 28th Annual Symposium on Foundation of Computer Science (1987)* pp. 78-87

[Russell-89] Russell, S.J. - Execution Architectures and Compilation, *Proceedings of the 11th IJCAI, Morgan Kaufmann 1989* pp. 15

[Schank-82] Schank R.C. - Dynamic Memory - A Theory of Reminding and Learning in Computers and People, *Cambridge University Press*, 1982

[Schapire-88] Schapire R.E. - Diversity-Based Inference of Finite Automata, *CS master thesis, M.I.T* 1988

[Searle-80] Searle J. - Minds, Brains and Programs, *Behavioral and Brain Sciences 3*(1980) pp 417-458

[Shapiro-81] Shapiro E.Y. - Inductive Inference of Theories From Facts, *Yale University, CS Dept. Research Report 192* (1981)

[Shavlik-89] Shavlik J.W. - Acquiring Recursive Concepts with Explanation-Based Learning, *Proceedings of the 11th IJCAI, Morgan Kaufmann 1989* pp 688

[Shavlik-90] Shavlik J.W. - Acquiring Recursive Concepts with Explanation-Based Learning, *Machine Learning Vol. 5 No. 1* (1990)

[Shen-90] Shen W. - Functional Transformations in AI Discovery Systems, *Artificial Intelligence 41* (1989/90) pp 257-272

[Smith-1982] Smith D.R. - A Survey of the Synthesis of LISP Programs from Examples, in Biermann, A.W. & Guiho, G. & Kondratoff, Y. (eds.) *Automatic Program Construction Techniques*, Macmillan Publ. Co. 1984

[Solomonoff-64] Solomonoff R.J. - A formal Theory of Inductive Inference, *Information and Control Vol 7* (1964) pp 1 (part I), pp 224 (part II)

[Solomonoff-78] Solomonoff R.J. - Complexity-Based Induction Systems: Comparisons and Convergence Theorems, *IEEE Transactions on Information Theory, Vol IT-24, No.4* (July 1978) pp 424-432

[Solomonoff-86] Solomonoff R.J. - The Application of Algorithmic Probability to Problems of Artificial Intelligence, in L.N.Kanal, Lemmer J.F. (eds.) *Uncertainty in Artificial Intelligence* (North Holland 1986)

[Solomonoff-90] Solomonoff R.J. - A system for Incremental Learning Based on Algorithmic Probability, in E. Pednault (ed.) *Symposium on The Theory and Applications of Minimal-Length Encoding* (1990) (Preprint of Proceedings)

[Stepp-86] Stepp R.E. III & Michalski R.S. - Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects, in Michalski, Carbonell, Mitchell (eds.): *Machine Learning* (vol2), Morgan Kaufmann 1986

[Ullman-85] Ullman S. - Visual Routines, in Pinker S. (ed) : *Visual Cognition*

[Utgoff-84] Utgoff P.E. - Shift of Bias for Inductive Concept Learning, *Ph.D. dissertation, Dept. of C.S. Rutgers University, 1984*

[Valiant-84] Valiant L.G. - A Theory of the Learnable, *Communications of the A.C.M. vol 27 no 11* (1984) pp 1134

[Van Gelder-90] Van Gelder, T. - Compositionality: A Connectionist Variation on a Classical Theme, *Cognitive Science 14*, (1990) 355-384

[Winston-80] Winston, P.H. - Learning and Reasoning by Analogy, *Communications of the ACM vol 23* (1980) no12

[Wolff-82] Wolff J.G. - Language Acquisition, Data Compression and Generalization: A Unifying View, *Language and Communication* 2 (1982) pp 57-89

[Wolf-87] Wolff J.G. - Cognitive Development as Optimization, in *L.Bolc (ed.): Computational Models of Learning* (1987) Springer Verlag

[Zhang-89] Zhang J. & Michalski R. - A Description of the Preference Criterion in Constructive Learning: A Discussion of Basic Issues, *Proceedings of the 6th Intl. Machine Learning Workshop* (1989) pp. 17

[Zvonkin-71] Zvonkin A.K. & Levin L.A. - The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms, *Russian Mathematical Surveys* Vol 25 (1971) pp 83

C-2

(2)

DATE DUE	BORROWER'S NAME

LIBRARY
N.Y.U. Courant Institute of
Mathematical Sciences

[illegible]

